Microprocessors

Tuba Ayhan

MEF University

ARM Processor – Addressing modes

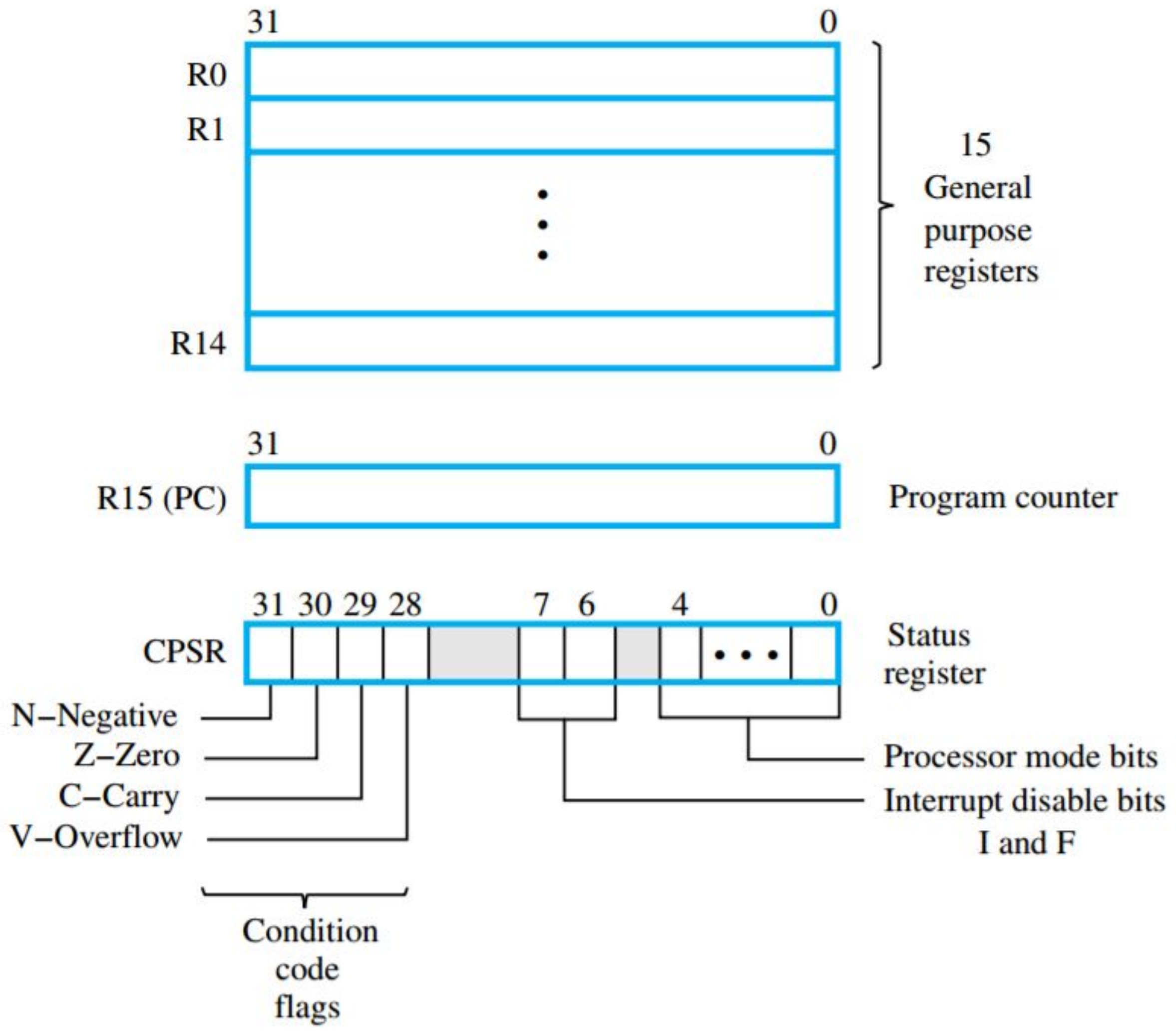
Computer Organization and Embedded Systems, Hamacher et. al

ARM Characteristics

- 32 bits word length, byte-addressable using 32-bit addresses memory, 32 bits long processor registers.
- 7 processor modes of operation. Application programs run in **User mode**. The other six modes are used to handle I/O device interrupts, processor powerup/reset, software interrupts, and memory access violations.
- RISC-style Aspects:
    - All instructions have a fixed length of 32 bits.
    - Only Load and Store instructions access memory.
    - All arithmetic and logic instructions operate on operands in processor registers
- CISC-style Aspects:
    - Autoincrement, Autodecrement, and PC-relative addressing modes are provided.
    - Condition codes (N, Z, V, and C) are used for branching and for conditional execution of instructions.
    - Multiple registers can be loaded from a block of consecutive memory words, or stored in a block, using a single instruction.

Register Structure

- 15 general-purpose registers: R0 - R14
- Program Counter (PC): R15.
- R13 and R14 are dedicated to the management of the processor stack and subroutines.
- The Current Program Status Register (CPSR), *the Status register*, holds the condition code flags (N, Z, C, V), interrupt-disable bits, and processor mode bits.
- **Banked registers** are duplicates of some of the R0 to R14 registers, to save and restore some User-mode register contents on mode switches.



Condition bits

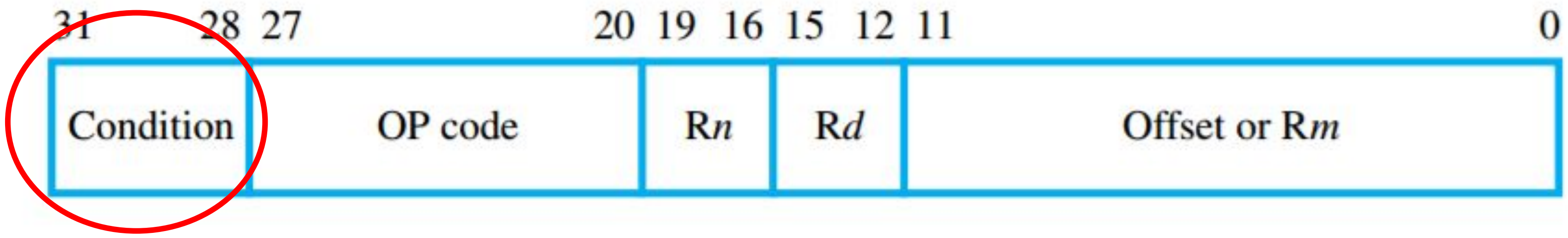
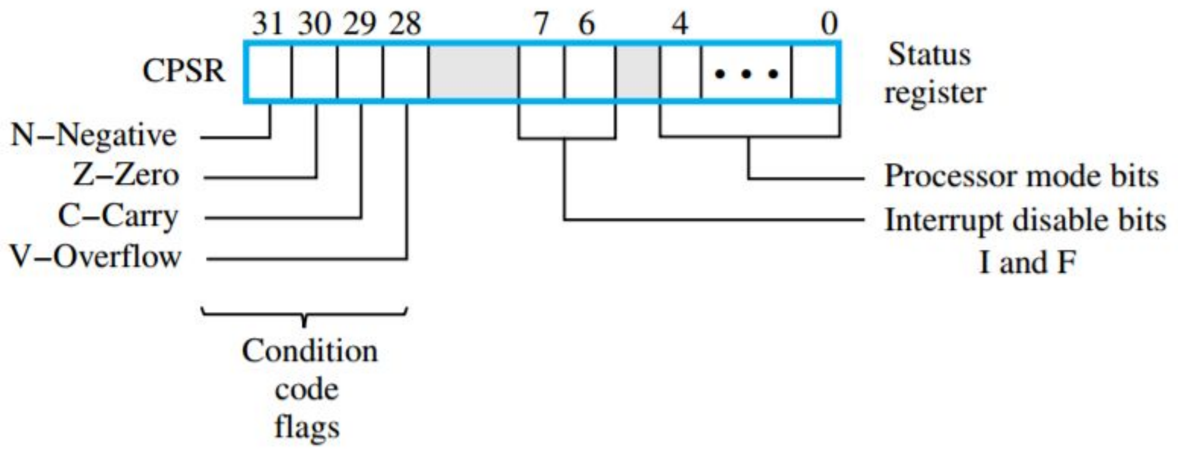


Figure D.2 Format for Load and Store instructions.

- The high-order four bits specify a condition that determines whether or not the instruction is executed. This is a special feature for ARM.
  - An instruction is executed only if the current values of the condition code flags satisfy the condition specified in a 4-bit field of the instruction.
  - Otherwise, the processor proceeds to the next instruction
  - One of the possible conditions always executed.

CPSR register

- Interrupt-disable bits, I and F, where
  - I = 1 disables the IRQ interrupts
  - F = 1 disables FIQ interrupts
- Thumb bit,
  - T = 0 indicates ARM execution
  - T = 1 indicates Thumb execution
- Processor mode bits which identify the mode in which the processor is operating.



Condition Code flags: based on the results of a previous operation.

- Negative (N) - set to 1 if the result is negative; otherwise, cleared to 0
- Zero (Z) - set to 1 if the result is 0; otherwise, cleared to 0.
- Carry (C) - set to 1 if a carry-out results from the operation; otherwise, cleared to 0.
- Overflow (V) - set to 1 if arithmetic overflow occurs; otherwise cleared to 0.

Addressing Modes

- Basic modes: The Immediate, Register, Absolute, Indirect, and Index addressing modes.
- More complex modes: Relative mode and variants of the Autoincrement and Autodecrement modes
- Many of these modes are derived from different forms of indexed addressing modes.

Register, Immediate, and Absolute Addressing Modes

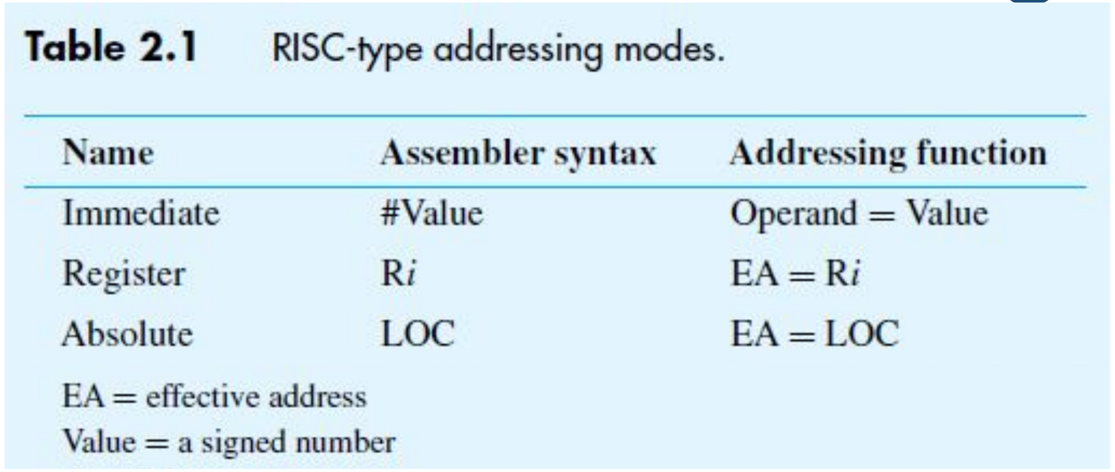


Table 2.1 RISC-type addressing modes.

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute	LOC	EA = LOC

EA = effective address
Value = a signed number

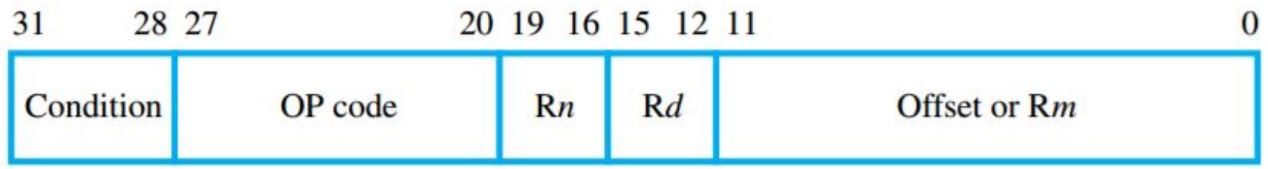


Figure D.2 Format for Load and Store instructions.

- The **Register addressing mode** is the main way for accessing operands in arithmetic and logic instructions. Constants are provided as 8-bit immediate values.
- **Immediate** addressing mode – 8-bit
- **Absolute** addressing mode -- 12-bit
- Advanced: The generation and use of 32-bit values as immediate operands or memory addresses.

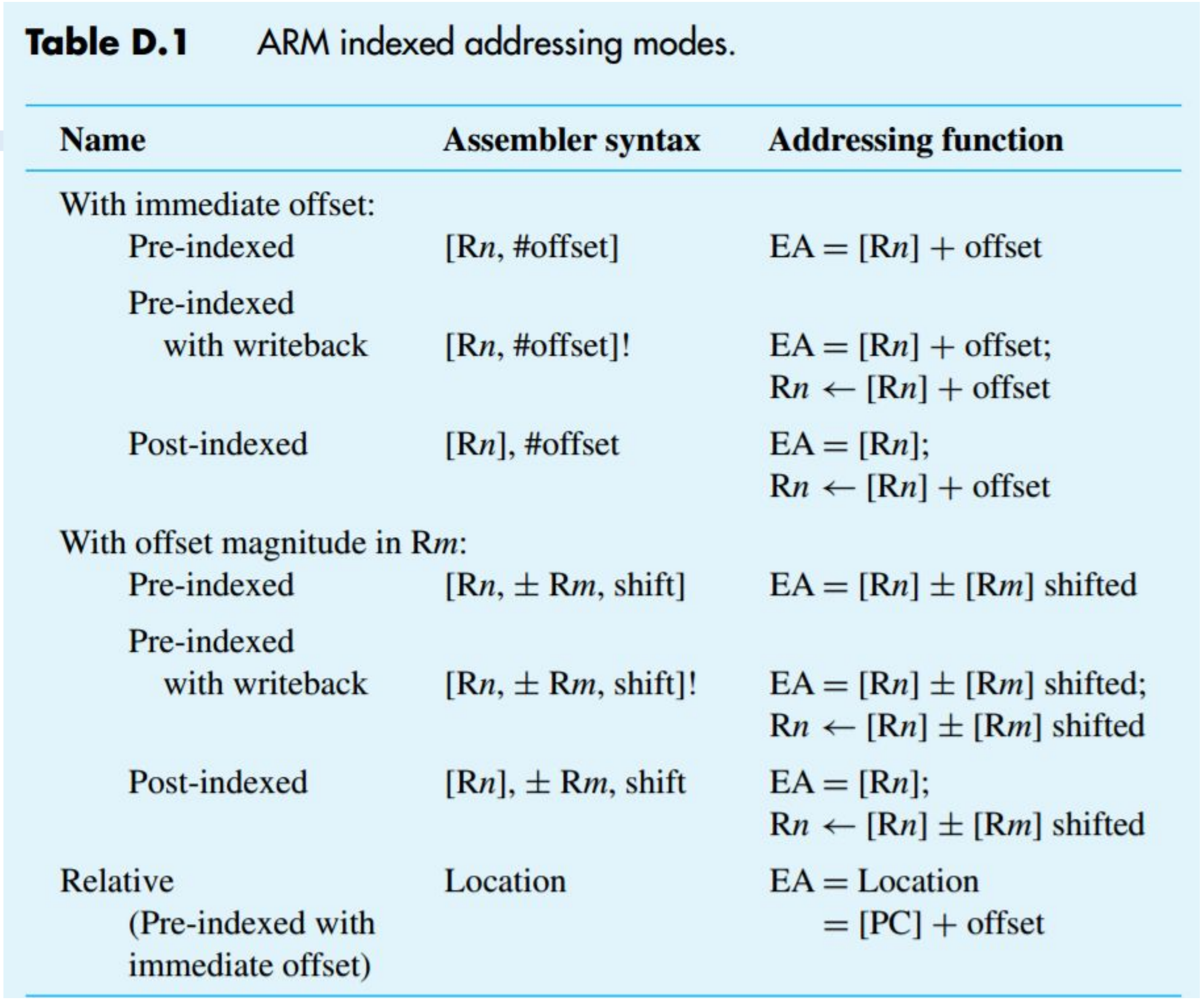


Table D.1 ARM indexed addressing modes.

Name	Assembler syntax	Addressing function
With immediate offset:		
Pre-indexed	[Rn, #offset]	EA = [Rn] + offset

Pre-indexed with writeback	[Rn, #offset]!	EA = [Rn] + offset;
		Rn ← [Rn] + offset

Post-indexed	[Rn], #offset	EA = [Rn];
		Rn ← [Rn] + offset

With offset magnitude in Rm:		
Pre-indexed	[Rn, ± Rm, shift]	EA = [Rn] ± [Rm] shifted

Pre-indexed with writeback	[Rn, ± Rm, shift]!	EA = [Rn] ± [Rm] shifted;
		Rn ← [Rn] ± [Rm] shifted

Post-indexed	[Rn], ± Rm, shift	EA = [Rn];
		Rn ← [Rn] ± [Rm] shifted

Relative (Pre-indexed with immediate offset)	Location	EA = Location = [PC] + offset

- **EA**: Effective address
- **Offset**: a signed number contained in the instruction
- **Shift**: direction # integer, direction is LSL for left shift or LSR for right shift, integer is 5-bit unsigned number.
- **Rm**: offset magnitude in Rm can be added to subtracted from the contents of the base register Rn.

Basic Indexed Addressing Mode

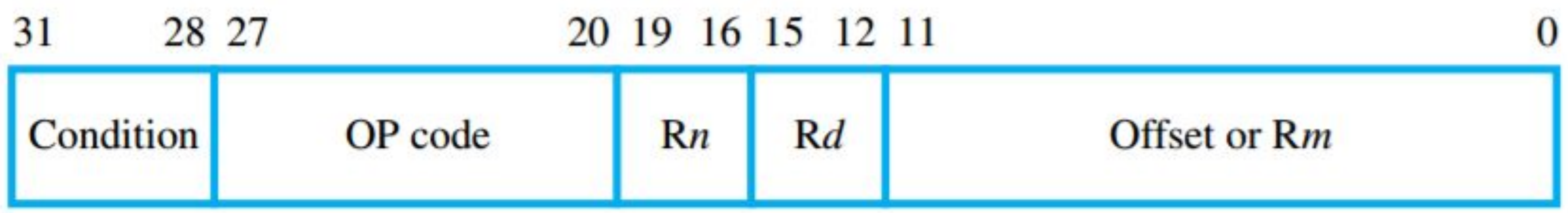


Figure D.2 Format for Load and Store instructions.

- *Pre-indexed mode* —The effective address of the operand is the sum of the contents of a base register, Rn, and a signed offset.

```
LDR Rd, [Rn]                 Rd ← [[Rn]]
LDR Rd, [Rn, #offset]        Rd ← [[Rn] + offset]
LDR Rd, [Rn, Rm]             Rd ← [[Rn] + [Rm]]
```

- The contents of Rm are the magnitude of the offset, Rm is preceded by a minus sign if a negative offset is desired.

Relative Addressing Mode

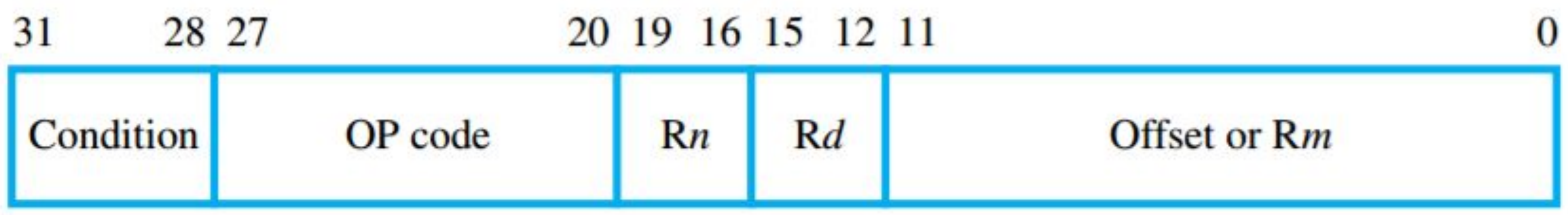


Figure D.2 Format for Load and Store instructions.

- PC is used as the Base register Rn, along with an immediate offset, in the Pre-indexed addressing mode.
- The programmer places the desired address label in the operand field to indicate this mode.

```
LDR R1, ITEM
```

- loads the contents of memory location ITEM into register R1.
- The assembler determines the immediate offset as the difference between the address of the operand and the contents of the updated PC. When the effective address is calculated at instruction execution time, the contents of the PC will have been updated to the address two words (8 bytes) forward from the instruction containing the Relative addressing mode.

Index Modes with Writeback

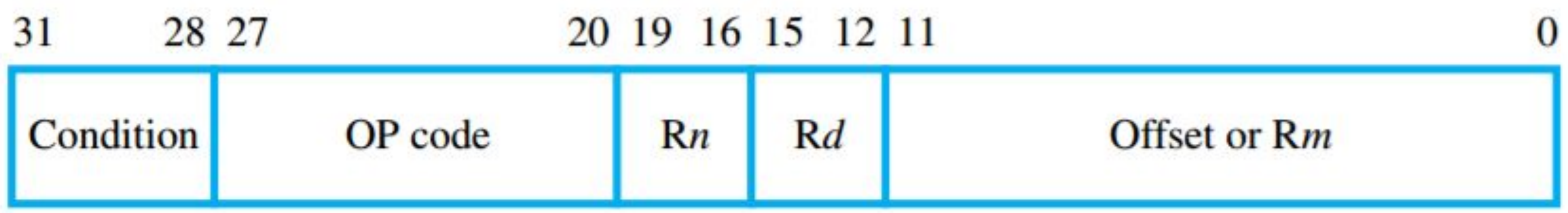


Figure D.2 Format for Load and Store instructions.

- *Pre-indexed with writeback mode* —The effective address of the operand is generated in the same way as in the Pre-indexed mode, then the effective address is written back into Rn. '!' signifies that writeback is to be done.

```
LDR Rd, [Rn, #offset]!            Rd ← [[Rn] + offset]
                                  Rn ← [Rn] + offset
LDR Rd, [Rn +-Rm,shift]!          Rd ← [[Rn] +- [Rm]shifted]
                                  Rn ← [Rn] +- [Rm]shifted
```

Post-indexed mode

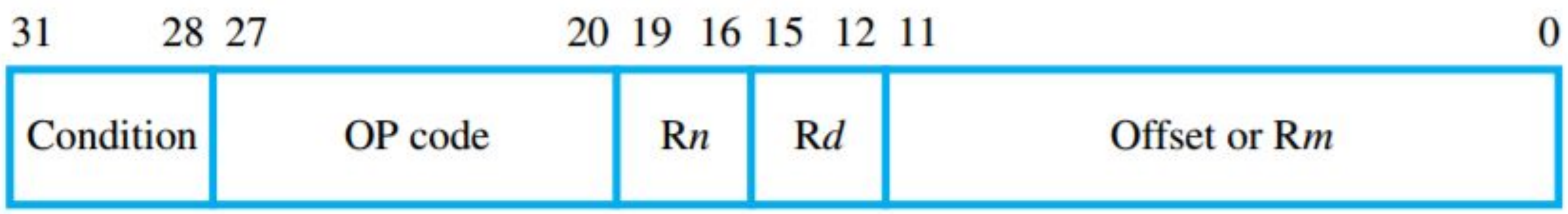


Figure D.2 Format for Load and Store instructions.

- *Post-indexed mode* —The effective address of the operand is the contents of Rn. The offset is then added to this address and the result is written back into Rn.

  LDR Rd, [Rn], #offset $\quad$ Rd ← [[Rn]]

  Rn ← [Rn] + offset

  LDR Rd, [Rn], +-Rm,shift $\quad$ Rd ← [[Rn]]

  Rn ← [Rn] +- [Rm]*shifted*

Square brackets

- When only the base register is enclosed in square brackets, its contents are used as the effective address. The offset is added to the register contents after the operand is accessed. post-indexing is specified. Ex:

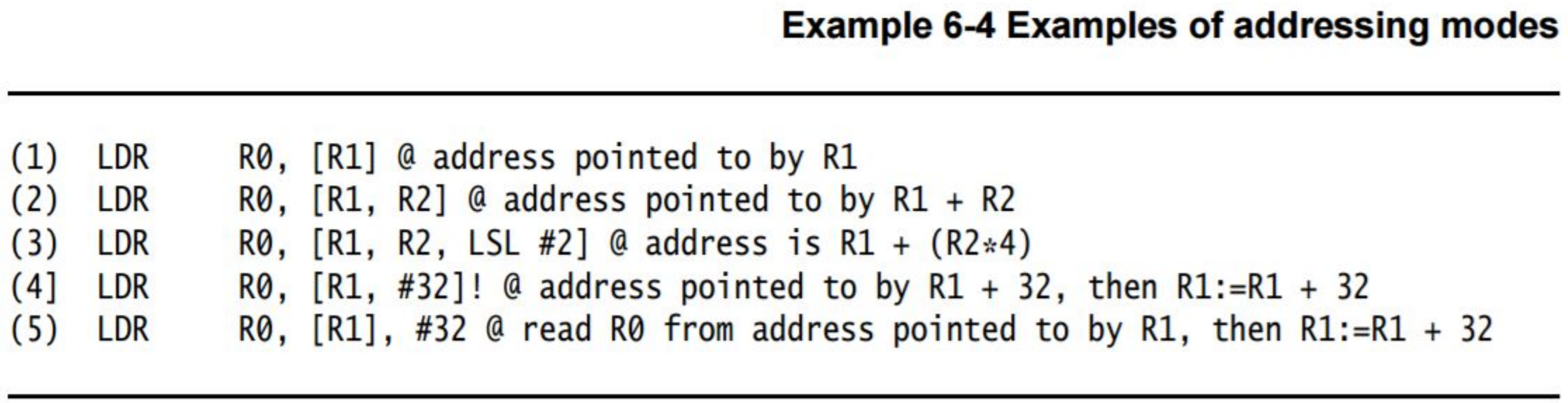
  LDR R1, [R2], *offset-is-here*

- When both the base register and the offset are placed inside the square brackets, their sum is used as the effective address of the operand, that is, pre-indexing is used. If writeback is to be performed, it must be indicated by the exclamation character.

  STR R1, [R2, *offset-is-here*]!

  STR R1, [R2, *offset-is-here*]

CortexA Programming guide - 6.4.1



Example 6-4 Examples of addressing modes

```
(1)  LDR     R0, [R1] @ address pointed to by R1
(2)  LDR     R0, [R1, R2] @ address pointed to by R1 + R2
(3)  LDR     R0, [R1, R2, LSL #2] @ address is R1 + (R2*4)
(4]  LDR     R0, [R1, #32]! @ address pointed to by R1 + 32, then R1:=R1 + 32
(5)  LDR     R0, [R1], #32 @ read R0 from address pointed to by R1, then R1:=R1 + 32
```

Offset Determination

1. An immediate value in the range ±4095.
2. Contents of the Rm register, with the sign (direction) of the offset specified by a ± prefix on the register name.

LDR R0, [R1, −R2]! $R0 \leftarrow [[R1] - [R2]]$

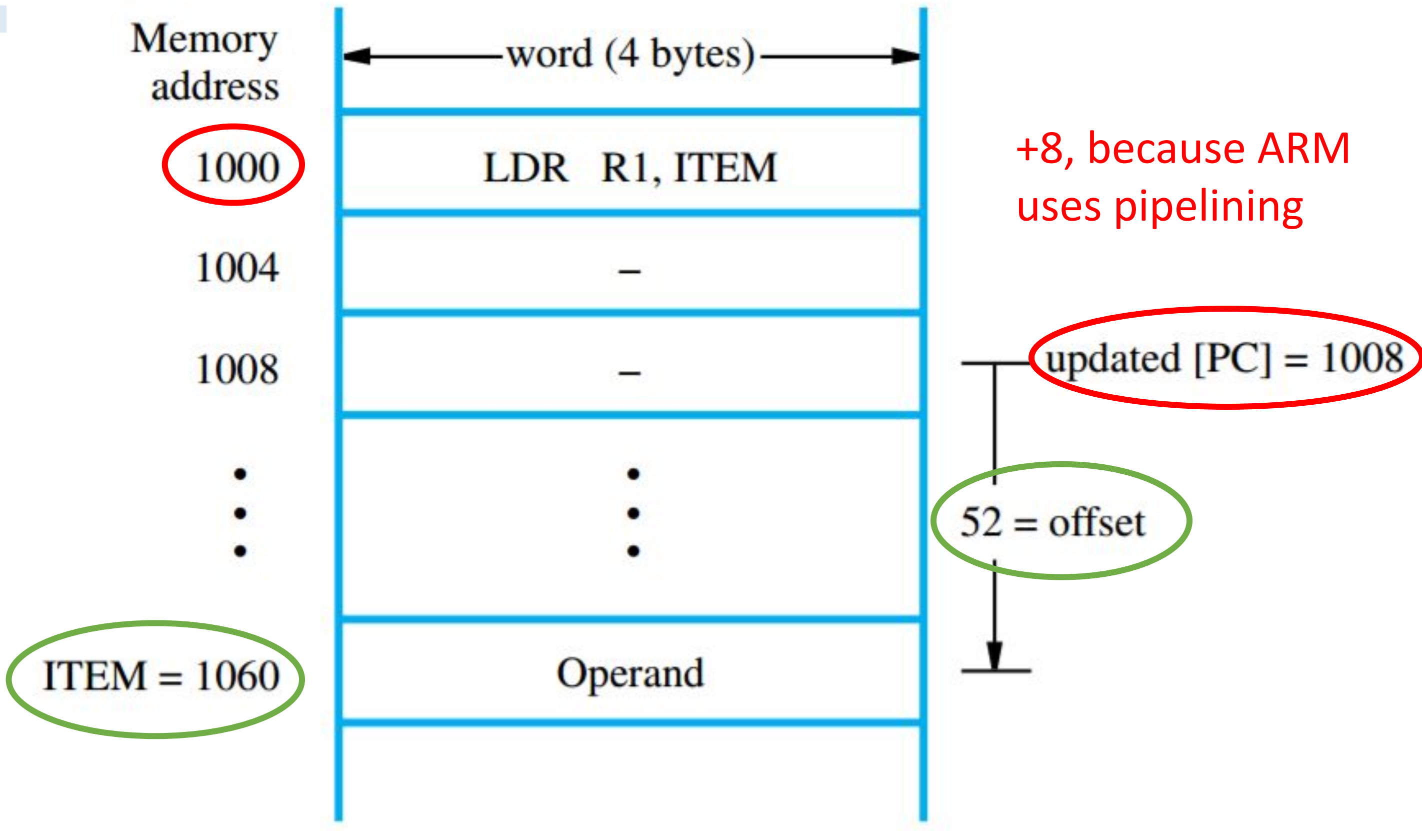
- Effective address of the operand: [R1] − [R2]
- Writeback is specified: EA is then loaded into R1.

LDR R0, [R1, −R2, LSL #4]! $R0 \leftarrow [[R1] - 16 \times [R2]]$

- The contents of R2 in the example above is multiplied by 16 before being used as an offset.

Example – Relative Addressing Mode

- The updated PC will contain 1008 when the offset is added to it during program execution.
- The offset calculated by the assembler is 52
- Effective address generated by this instruction is 1060 = 1008 + 52



+8, because ARM uses pipelining

The operand must be within a distance of 4095 bytes from the updated PC. If not, an error is indicated by the assembler and a different addressing mode must be used to access the operand.

Load the numbers on the red locations

Use R0 as base register.

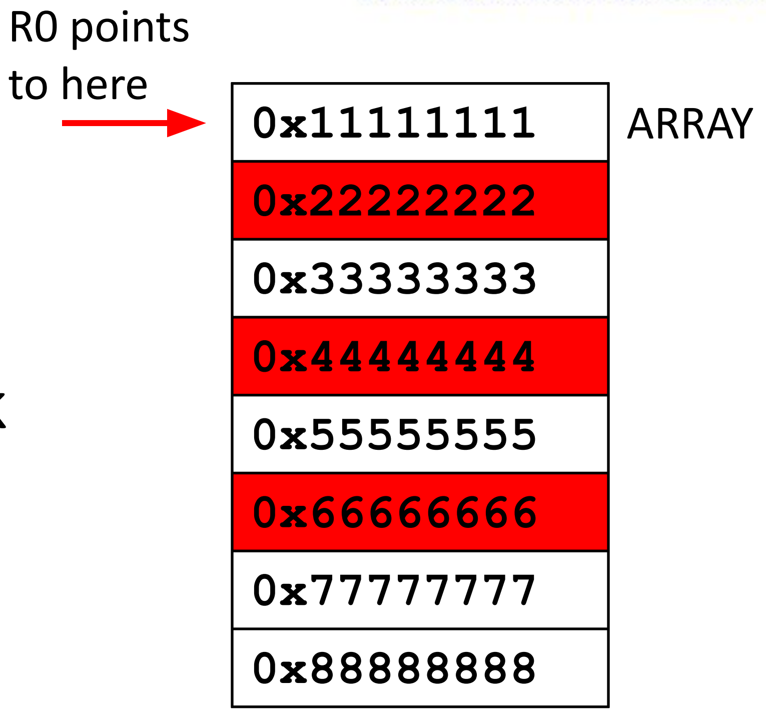
- Don’t use an offset.

```
ADD R0, R0,  #4
LDR R2, [R0]
ADD R0, R0,  #8
LDR R4, [R0]
ADD R0, R0,  #8
LDR R6, [R0]
```

- Post-index

- Pre-index

- Pre-index with write back



Load the numbers on the red locations

Use R0 as base register.

- Don't use an offset.

```
ADD R0, R0,  #4
LDR R2, [R0]
ADD R0, R0,  #8
LDR R4, [R0]
ADD R0, R0,  #8
LDR R6, [R0]
```

- Post-index

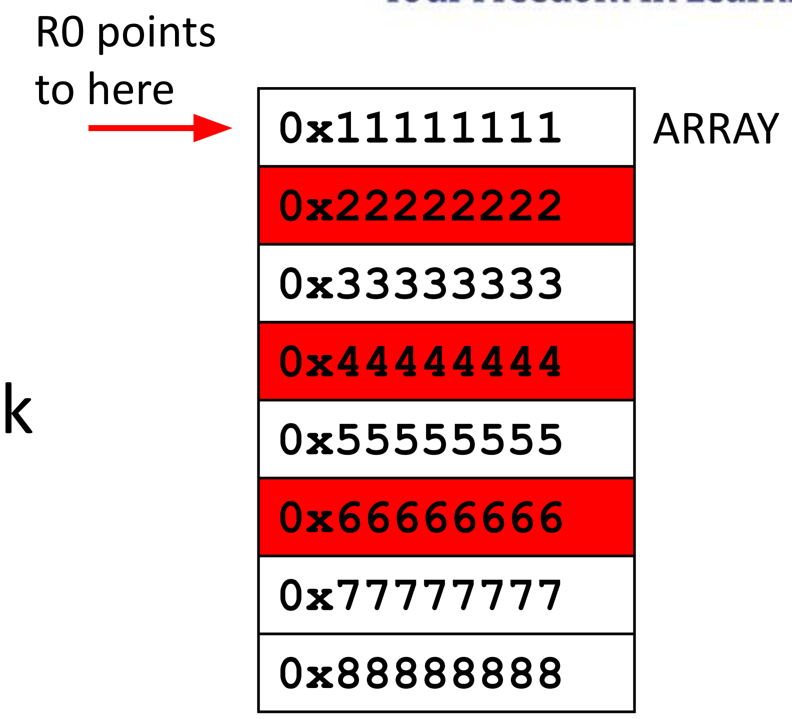
```
ADD R0, R0, #4
LDR R2, [R0], #8
LDR R4, [R0], #8
LDR R6, [R0]
```

- Pre-index

```
LDR R2, [R0, #4]
LDR R4, [R0, #12]
LDR R6, [R0, #20]
```

- Pre-index with write back

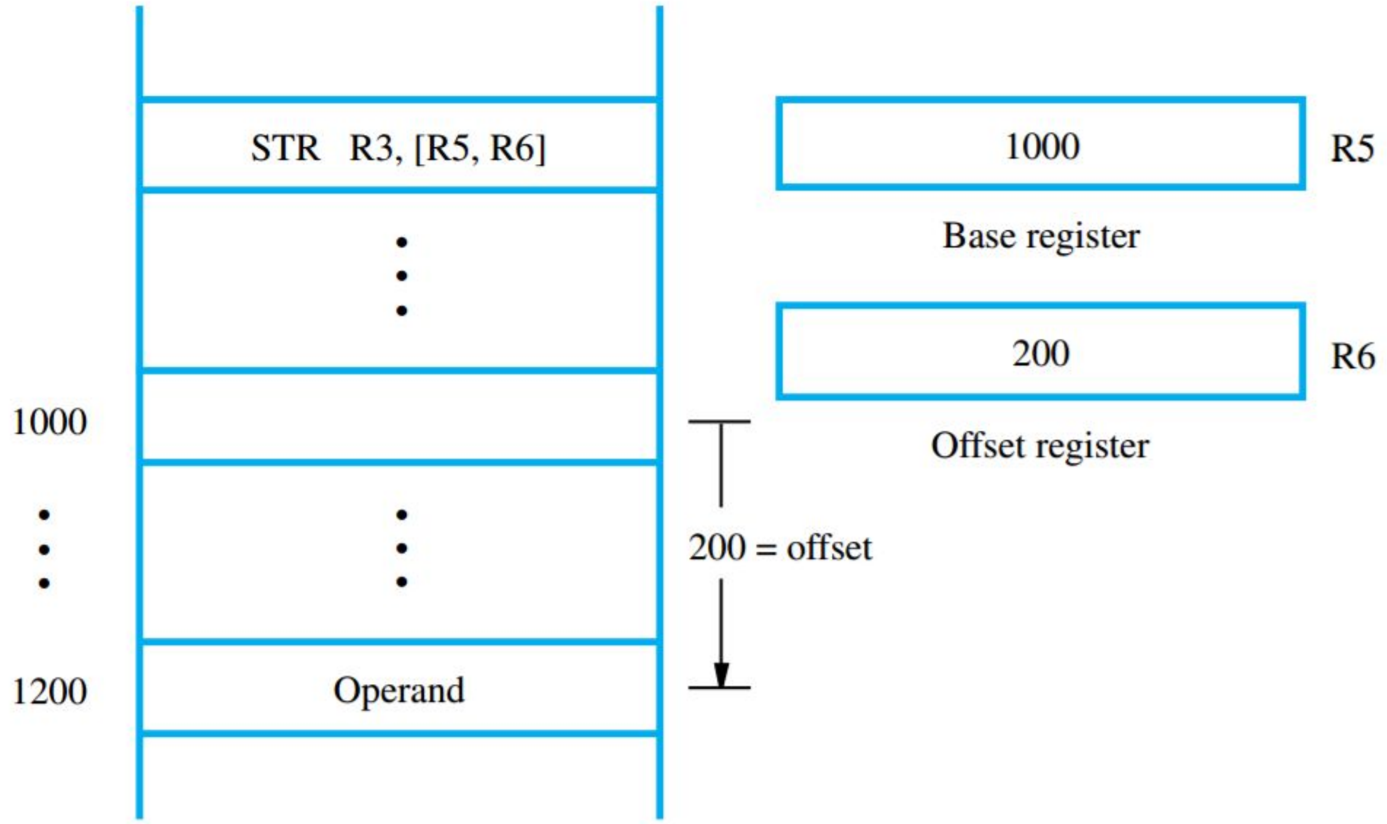
```
LDR R2, [R0, #4]!
LDR R4, [R0, #8]!
LDR R6, [R0, #8]
```



Rest of the slides are not covered on Feb 21. 2022.

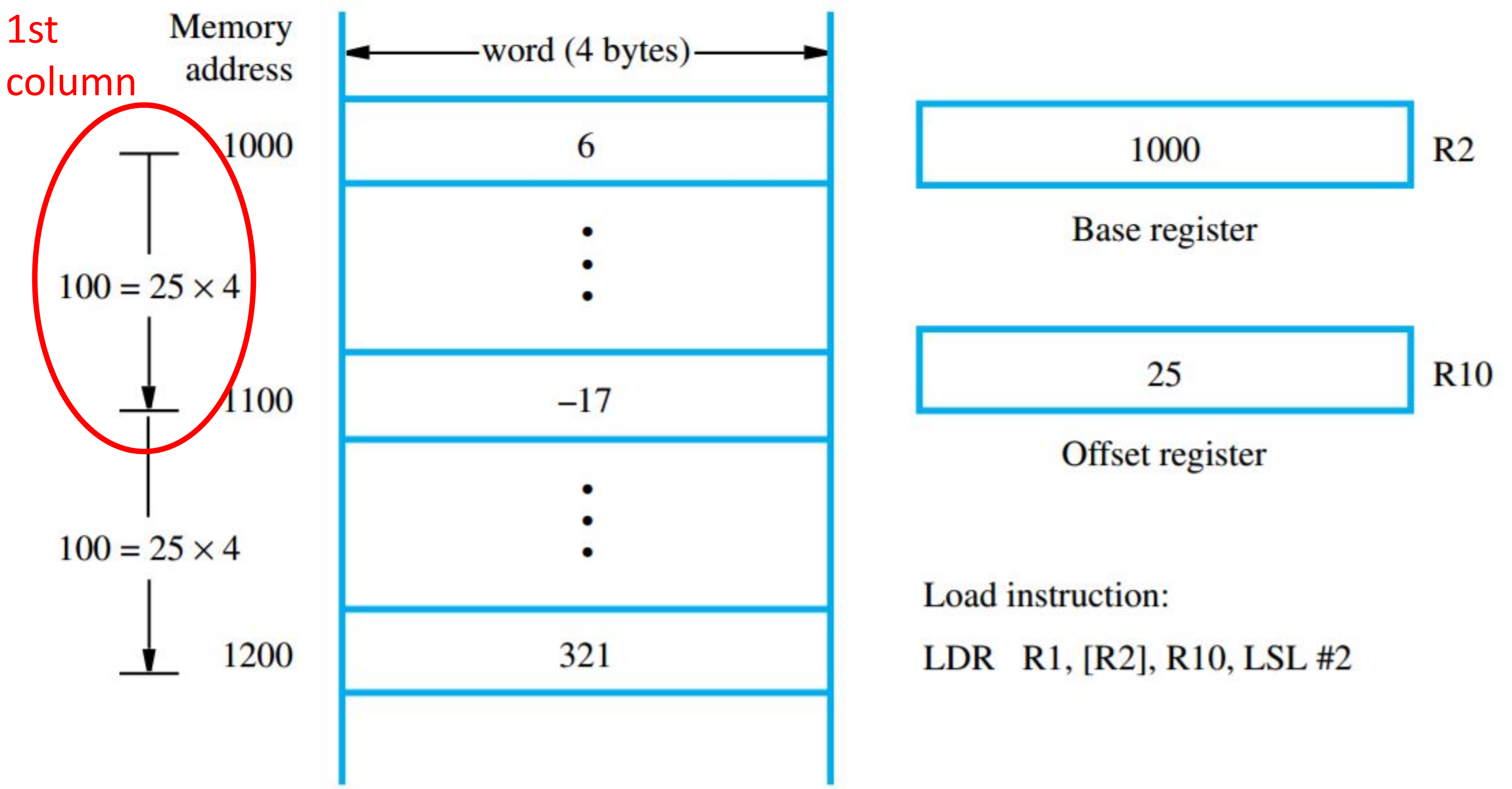
Example – Pre-indexed addressing mode

- The offset contained in register R6.
- The base value contained in R5.
- The Store instruction (STR) stores the contents of R3 into the word at memory location 1200.



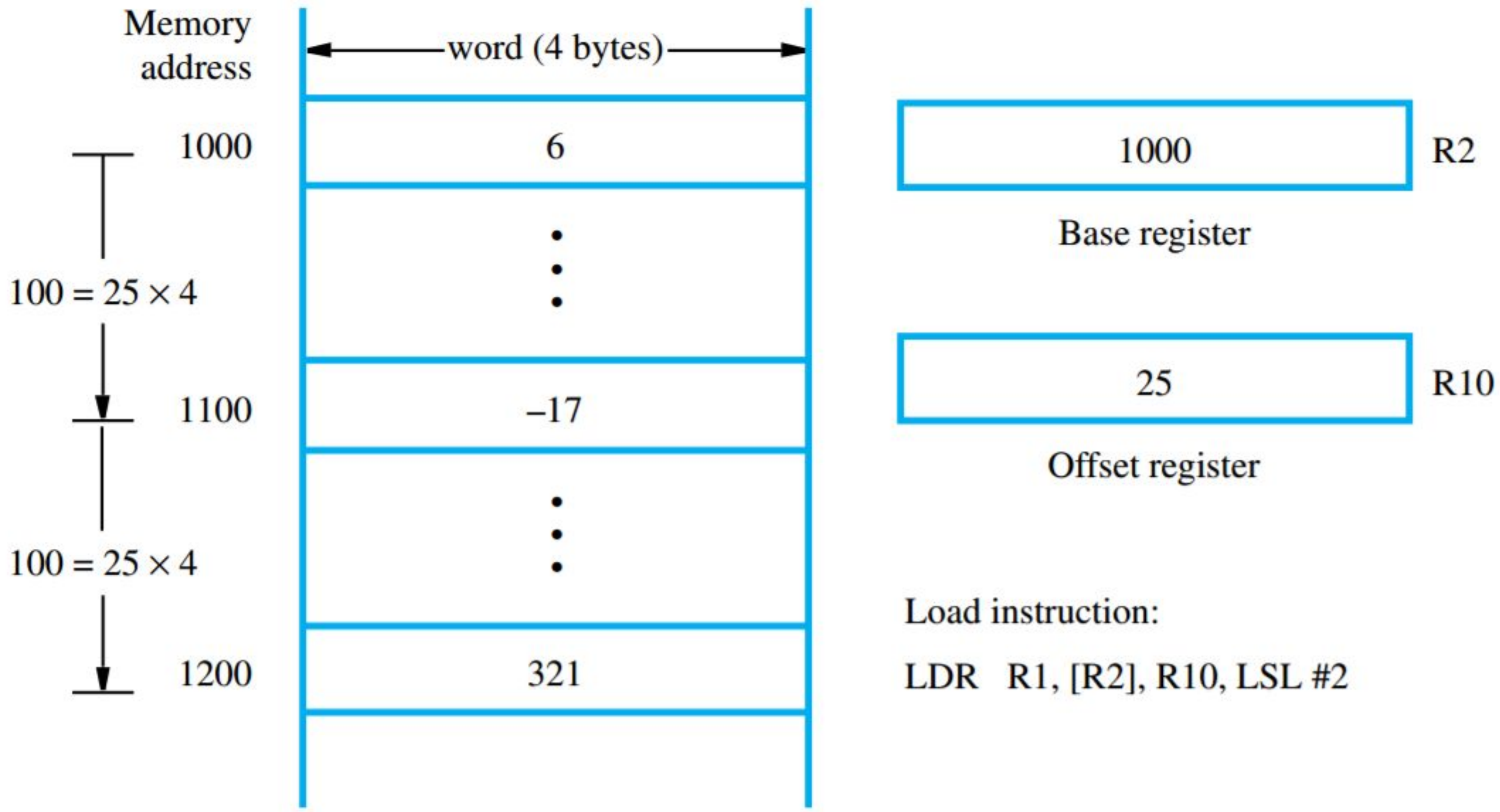
Example – Post-indexed Addressing

- A 25 × 25 matrix of numbers is stored in column order.
- Access the numbers in the first row of the matrix: 1000, 1100, 1200….



Example – Post-indexed Addressing

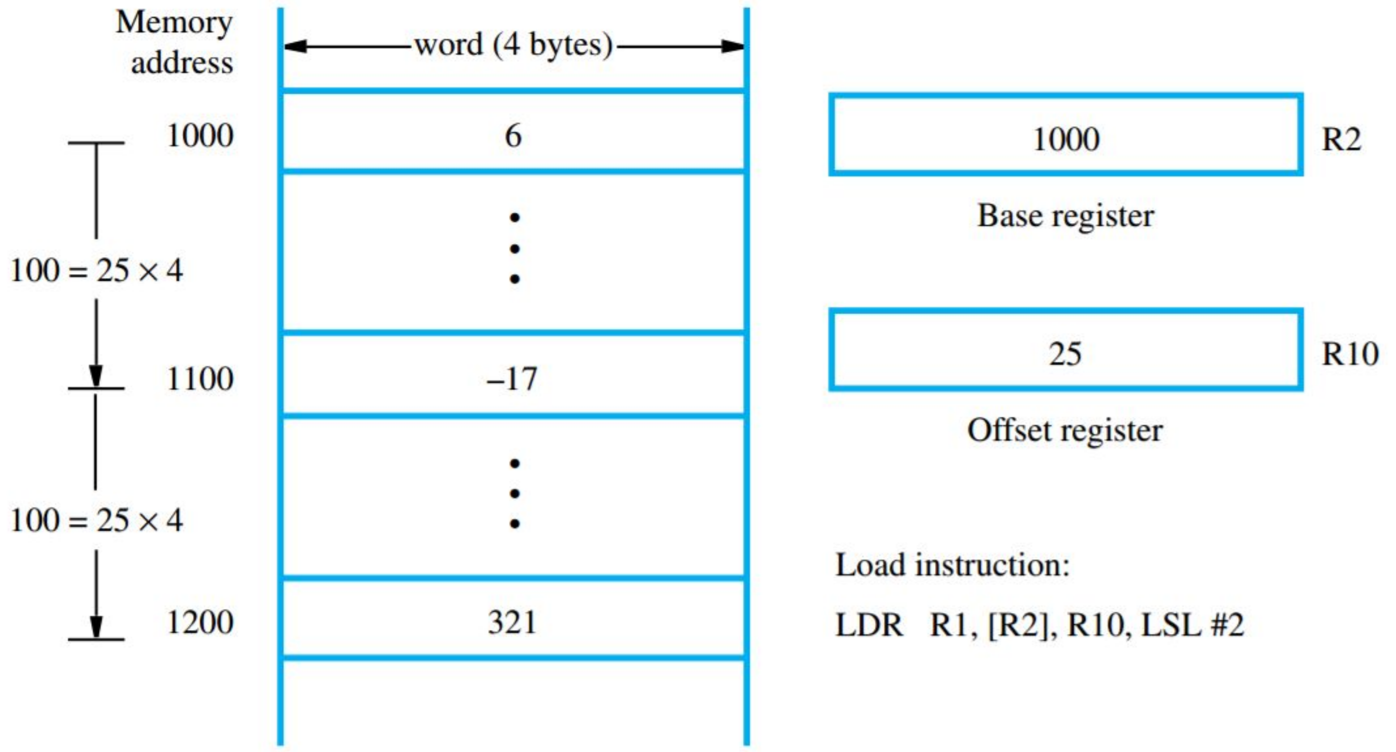
- R2 is used as the base register: contains the initial address value 1000.
- R10 is used to hold the offset = 25.
- R1 is used to load successive elements of the first row of the matrix in successive passes through the loop



Example – Post-indexed Addressing

LDR R1, [R2], R10, LSL #2

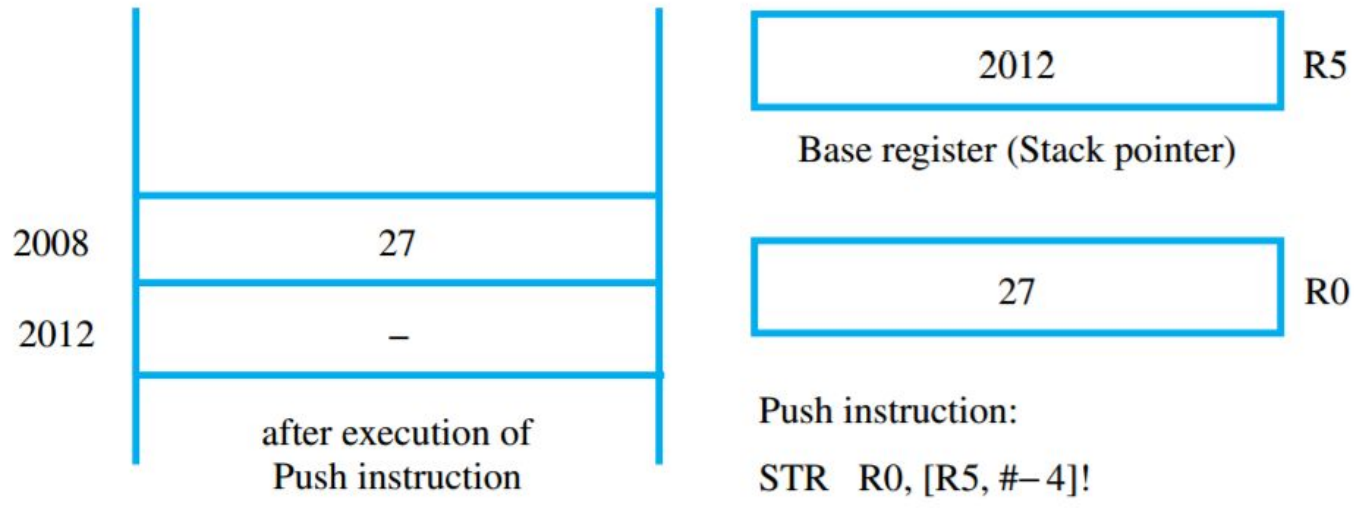
- First loop:
- EA is [R2] = 1000.
- The number 6 at address 1000 is loaded into R1.
- The writeback operation changes the contents of R2 from 1000 to 1100:
    - Contents of R10 is shifted right by 2 bits: 25x4 = 100.
    - Contents of R2 is increased by 100.
    - The contents of R10 are not changed.



Example – Pre-indexed Addressing

- Register R5 is used as the stack pointer.
- Initially R5: TOS (top-of-stack) element = 2012.
- The immediate offset −4 is added to the contents of R5 and the new value is written back into R5.
- EA of store operation = 2008.

STR R0, [R5, #−4]!


2008
27
2012
–
after execution of Push instruction
2012
R5
Base register (Stack pointer)
27
R0
Push instruction:
STR R0, [R5, #– 4]!