CZ1106
CE1106

# Chapter 3

# Instruction Organisation in Memory

©2020, SCSE/NTU

1

CZ1106
CE1106

Chapter 3

## Instruction Organisation in Memory

## Characteristics of Instructions and the ARM Programmer's Model

**Learning Objectives (3.1)**

1. Describe the characteristics and role of an instruction.
2. Describe the function of the ARM instruction set **MOV** instruction
3. Describe the ARM programmer's model.
4. Describe the functions and interpretation of several ARM registers.

©2020, SCSE/NTU

2

CPU only understand bit patterns of instructions

data is non-executable

CZ1106 CE1106

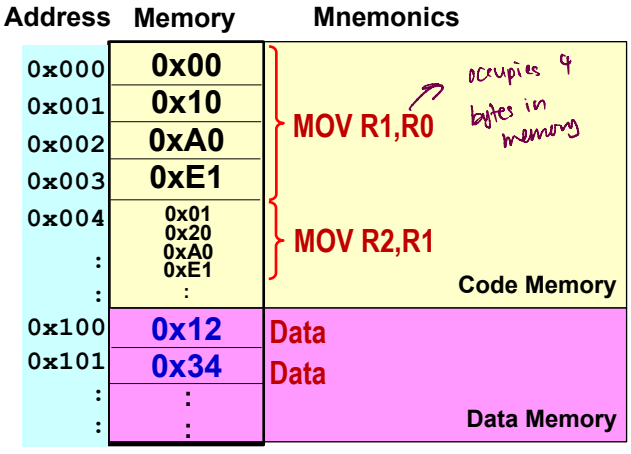
## Code and Data in Memory

- Instructions are stored in memory as binary patterns called **machine codes**.
- They are also represented in more readable **mnemonics**.
- ARM (32-bit CPU) example:

**Note:** ARM instructions are **32 bits** long

```
MOV R1,R0
MOV R2,R1
:
```

- Memory is partitioned into separate **code** and **data** segments.



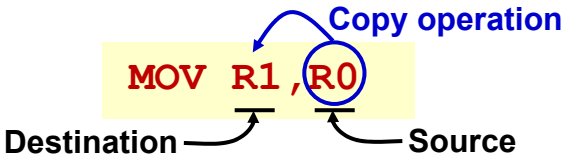

©2020 SCSE/NTU

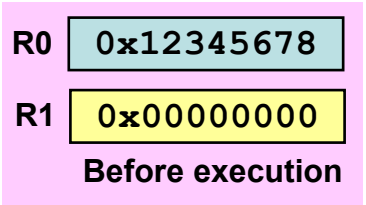
3

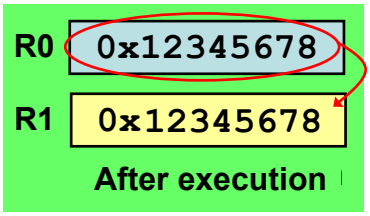
CZ1106 CE1106

## ARM Instruction Format (Introduction)

- Introduction to a simple ARM mnemonic.
- The **MOV** operator is a two-operand instruction that copies the source operand to the destination operand.
- The right operand is the source and left operand is the destination.




- In this example, the both operands are **registers** in the ARM CPU.

©2020 SCSE/NTU

4

CZ1106 CE1106

# Instruction Organization in Memory

- Instructions are binary encoded and stored in memory. Unlike data, instruction bytes tell CPU what actions to take (i.e. they are **executable**).
- Most instruction formats consist of **two** parts.


  - First part of instruction (**op-code**) specifies the operation to be carried out (e.g. move, add, subtract).
  - Remaining bits (**operands**) specify the data itself or the location of data and where results is to be stored.
  - Some operations produce no storable result but may alter program sequence or influence status flags (e.g. **N**, **Z**, **V**, **C**).

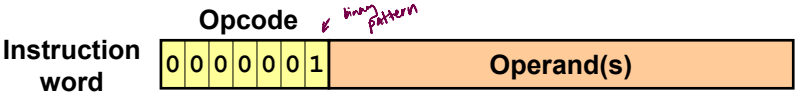
©2020, SCSE/NTU

5

CZ1106 CE1106

# Opcode and Operands

- **How are opcode encoded in an instruction?**




  - If there are 80 different operations, (e.g. **`add`**, **`sub`**, etc) then at least 7 bits ($2^6 < 80 < 2^7$) of opcode is needed to represent all the unique bit patterns.
  - The **more variety** of operations supported by the instruction set, the **longer the length** of each instruction.
- **Are there many ways to specify the operand(s)?**
  - Numerous. An operand can be stored as part of the **instruction**, stored in a **register** or stored in **memory**.
  - The method by which the operand is specified is known as **addressing mode**.

©2020, SCSE/NTU

6

CZ1106 CE1106

# Addressing Mode Examples

| Addressing Mode | ARM | Intel |
|---|---|---|
| Absolute (Direct) | None | `MOV AX,[1000h]` |
| Register Direct | `MOV R1,R0` | `MOV AX,DX` |
| Immediate | `MOV R1,#3` | `MOV AX,0003h` |
| Register Indirect | `LDR R1,[R0]` | `MOV AX,[BX]` |
| Register Indirect with Offset | `LDR R1,[R0,#4]` | `MOV AX,[BX+4]` |
| Register Indirect with Index | `LDR R1,[R0,R2]` | `MOV AH,[BX+DI]` |
| Implied | `BNE LOOP` | `JMP -8` |

©2020 SCSE/NTU

7

CZ1106 CE1106

# Role of Instructions

- The role of an instruction is to make purposeful **changes** to the **current state** of the processor.
  - The visible current state of a processor is defined by the **programmer's model** and contents in **memory**.
  - There are three broad categories of instructions for general programming available in a processor:

lets us make jumps, decisions

**Data Processing**

ARM examples:
```
ADD R0,R1,R2
SUB R1,R2,#3
EOR R3,R3,R2
```

**Data Transfer**

ARM examples:
```
MOV R1,R0
STR R0,[R2,#4]
LDR R1,[R2]
```

**Program Control**

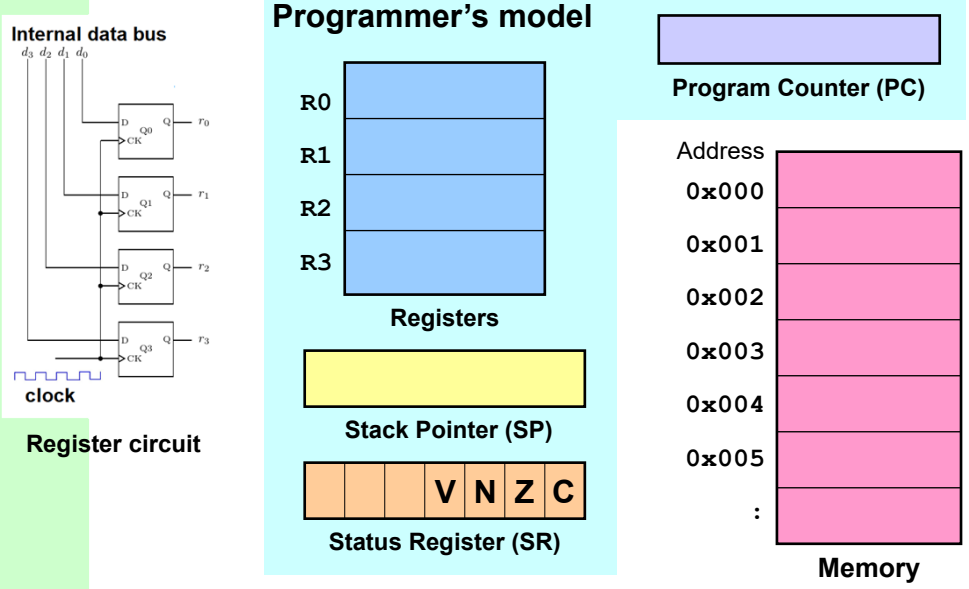
ARM examples:
```
B   Back
BNE Loop
BL  Routine
```

©2020 SCSE/NTU

8

CZ1106 CE1106

## Typical Programmer's Model & Memory




©2020 SCSE/NTU

9

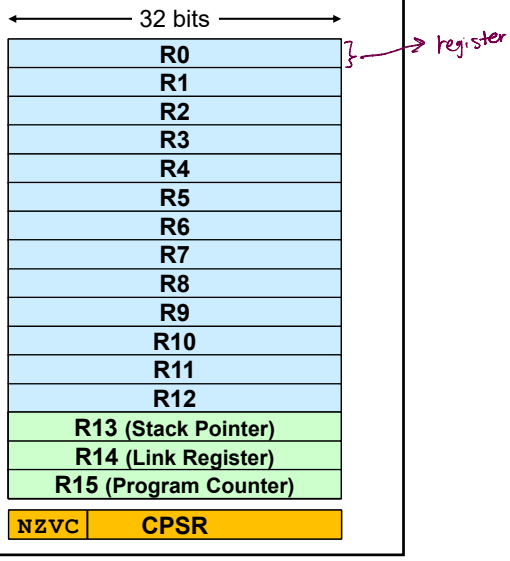
PC: manage by CPU auto point to next instruction to be executed

CZ1106 CE1106

## The ARM Programmer's Model

- The ARM processor can operate in various modes. The following are visible registers in the **User mode**.
  - There are 16 32-bit registers.
  - R0 – R12 are general purpose registers.
  - R13 is the Stack Pointer (**SP**).
  - R14 is the Link Register (**LR**).
  - R15 is the Program Counter (**PC**)
  - The Current Processor Status Register (**CPSR**) holds the condition code bits (**NZVC**)

Special function registers




©2020 SCSE/NTU

10

Instruction execution may involve multiple accesses to memory, resulting in different number of clock cycles to completely execute the instruction.

To start executing program:
get program counter to point to address 000H to start for instruction

each instruction is 4 bytes in ARM
⇒ requires a start address to allow to retrieve the consecutive 4 bytes

CZ1106 CE1106

## Program Counter

- ARM's register **R15** in the is the Program Counter (**PC**) and it keeps track of **program execution.**
  - The content of the **PC** is an **address**.
  - This address is the start address of the **next instruction** to be fetched.
  - The **PC automatically increments** by the length of the instruction executed (i.e. increment by 4 in the ARM CPU).
  - Sequential execution can be altered by modifying the contents of the **PC** to a new address location (i.e. a jump or a branch operation)
  - Due to **pipeline** architecture of the ARM CPU, the value in the **PC** is the address of the current instruction being executed plus 8 bytes (i.e. 2 instructions).

©2020 SCSE/NTU

11

CZ1106 CE1106

## Stack Pointer and Link Register

- Stack Pointer (**SP**).
  - By convention, **R13** in the ARM processor is designated the stack pointer.
  - The **SP** is used to maintain a space in memory **(stack)** that is used to **temporarily** stored away register information which will be needed again later.
- Link Register (**LR**).
  - **R14** is used as a Link Register during the calling of subroutines.
  - **R14** gets a copy of the **PC** (R15) when a Branch with Link (**BL**) instruction is executed.
  - At all other times, **R14** can also act as a general purpose register.

**Learn more:** Google Search "ARM stack pointer" or "Link Register"

©2020 SCSE/NTU

12

CZ1106 CE1106

# CPSR Condition Code Flags

| CPSR bit(s) | Name | Description (NZVC CPSR) |
|---|---|---|
| 31 | N | Can set if last operation produced a **negative** result |
| 30 | Z | Can set if last operation produced a **zero** result |
| 29 | C | Can set if last operation produced a **carry out** in the most significant bit of your 32 bit value ⇒ signifies overflow in unsigned number |
| 28 | V | Can set if the last operation produced an **overflow** for a **signed** arithmetic operation → when result cannot be represented |

**N** – Negative; **Z** – Zero; **C** – Carry ; **V** – Overflow

©2020 SCSE/NTU

13

CZ1106 CE1106

# Summary

- An instruction usually consist of an **opcode** and an **operand**.
- Instructions in a program change the **states** of a processor as it executes.
- The states of the processor consist of values in:
  - general purpose registers (e.g. **R0 – R12**)
  - specialised function registers (e.g. **CPSR, SP**, **LR** and **PC**)
  - memory contents
- The design of a program is to ensure that these **states changes** in a purposeful manner during the execution of the sequence of instructions.

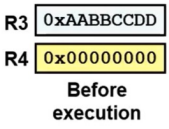
©2020 SCSE/NTU

14

CZ1106
CE1106

# Quiz: The MOV Instruction

- Based on the initial state shown, which option is the correct register state after executing the ARM mnemonic `MOV R3,R4`?





a. R3 0x00000000, R4 0x00000000 — After execution

b. R3 0xAABBCCDD, R4 0xAABBCCDD — After execution

c. R3 0xAABBCCDD, R4 0x00000000 — After execution

d. R3 0x00000000, R4 0xAABBCCDD — After execution

©2020. SCSE/NTU

CZ1106
CE1106

# Quiz: ARM Programmer's Model

- Which of the following is(are) **not** valid register(s) in the ARM programmer's model?

  Note: You can indicate more than one selection.

ARM has 16 32-bit registers, **R0** to **R15**. As such, **R16** is not a valid register. **R13**, **R14** and **R15** are also known as the **SP**, **LR** and **PC** respectively. The overflow **V** flag is not a register but a condition code bit (or flag) in the CPSR. It indicates if an overflow has occurred during a signed arithmetic operation.

a. R16

b. PC = R15

c. LR = R14

d. V

e. SP = R13

CZ1106
CE1106

Chapter 3

# Instruction Organisation in Memory

## Basic Execution Cycle

**Learning Objectives (3.2)**

1. Describe the function of a simple **LDR** instruction from the ARM instruction set.
2. Describe the basic execution cycle of a simple **LDR** instruction.
3. Describe the factor that determines the execution speed of an instruction.

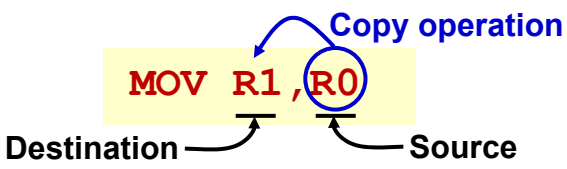
©2020 SCSE/NTU

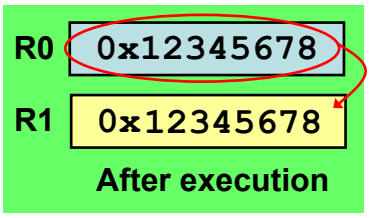
15

CZ1106
CE1106

## The MOV Instruction (Review)

- The `MOV` operator copies **register** content to a register.




©2020 SCSE/NTU

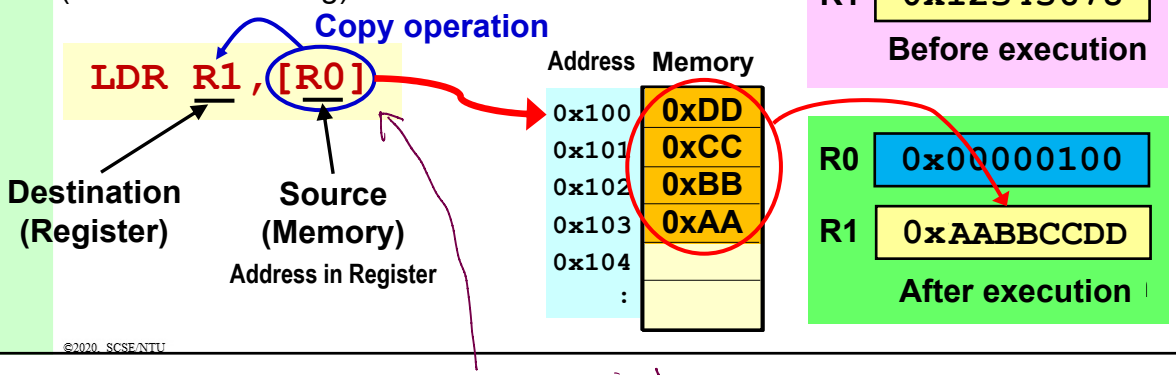
16

MOVS :

CZ1106 CE1106

# The LDR Instruction

- The `LDR` operator copies **memory** content to a register.
  - The left operand is always a destination register.
  - The right source operand is a memory location whose address is contained in a register (indirect addressing).




©2020 SCSE/NTU

17

instead of just copying ... 100 it points to the address 0x100 and copies the value of the address to R1

CZ1106 CE1106

# The Role of the Processor

- What is the role of the processor (CPU)?
  - **Fetch instructions** - CPU reads instructions from memory.
  - **Decode instructions** - Instruction is decoded to determine action required. → bit patterns
  - **Fetch data** – Some data from memory may need to be fetched in order to complete execution of instruction (optional).
  - **Execute** – Instruction execution may require CPU to perform some arithmetic or logical operation on the data, or just move the data into a register.
  - This **fetch-decode-execute** cycle is performed repeatedly by the CPU once powered-on.

©2020 SCSE/NTU

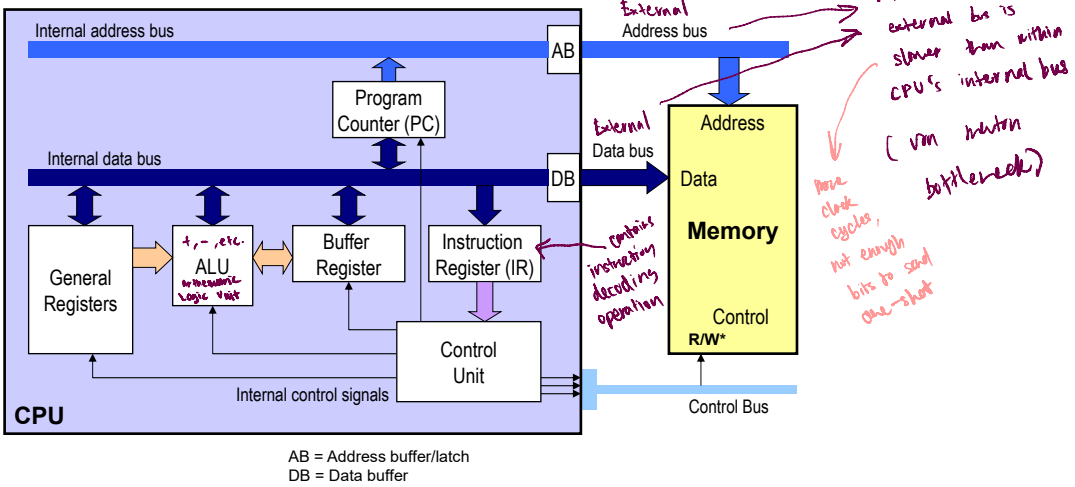
18

CZ1106
CE1106

## Basic Execution Cycle

# A Simple Processor

- Basic components of a simple processor (CPU) and its interface signals to external main memory.




©2020, SCSE/NTU

19

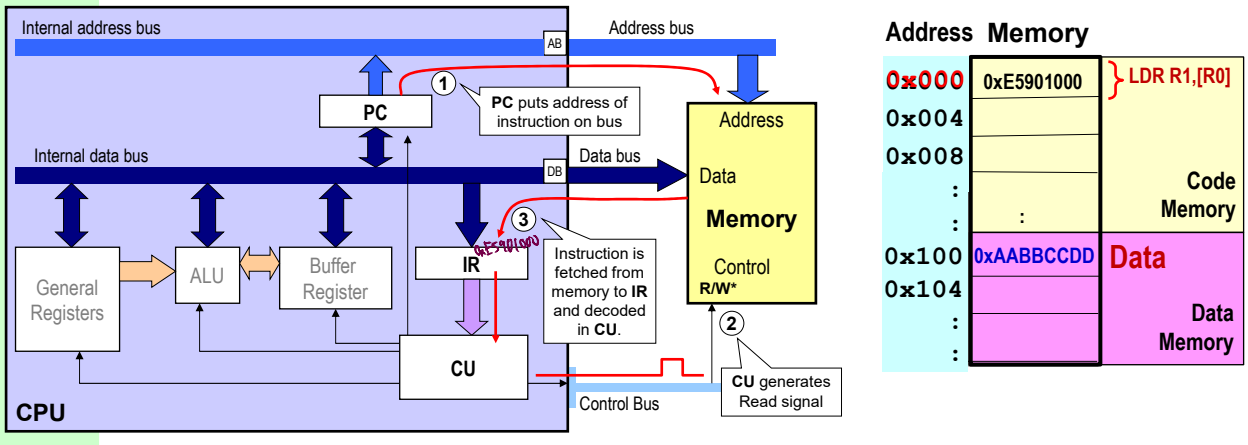
Instruction execution may involve multiple accesses to memory, resulting in different number of clock cycles to completely execute the instruction.

CZ1106
CE1106

## Basic Execution Cycle

# Fetch Cycle - Instruction

- Consider an instruction like: `LDR R1,[R0]`.
  In the fetch cycle, the **PC** points to address of next the instruction and its opcode is fetched into the **IR**.




©2020, SCSE/NTU

20

CZ1106
CE1106

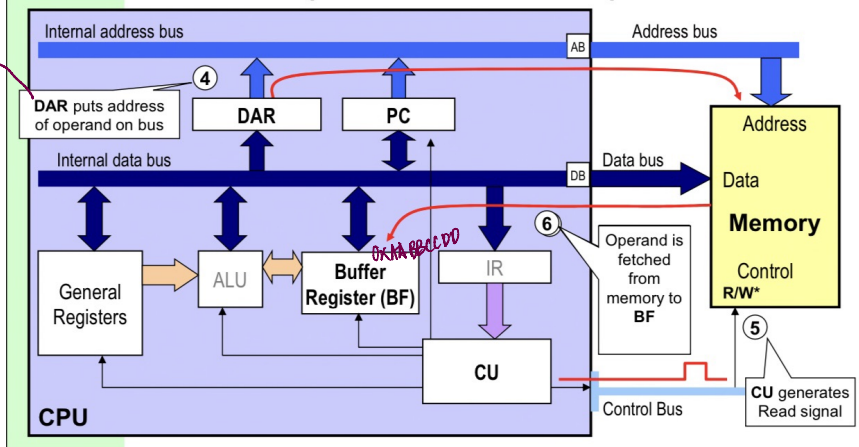
## Basic Execution Cycle

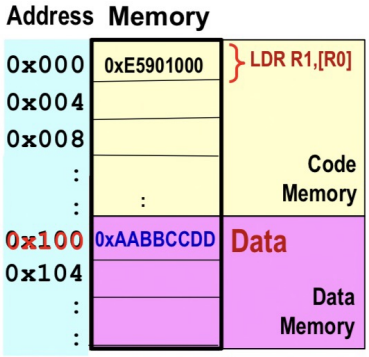
# Fetch Cycle - Operand

DAR used to fetch from memory, not the PC

- Decoding `LDR R1,[R0]` suggest an operand is needed. Since its is stored in memory, another fetch is required to retrieve operand into CPU.

Data Address Register




©2020 SCSE/NTU

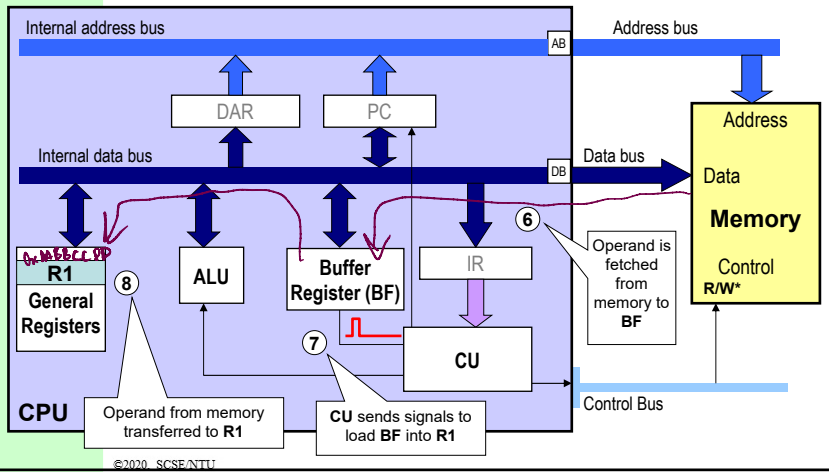
21

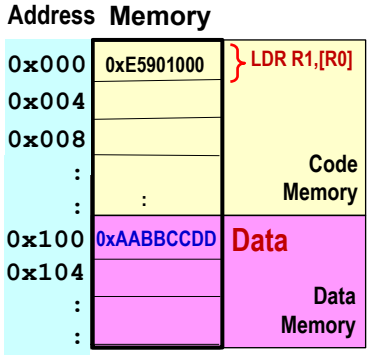
CZ1106
CE1106

## Basic Execution Cycle

# Execute Cycle - Load

- In the execution cycle of `LDR R1,[R0]`, the operand fetch from memory is loaded into the destination register **R1**.



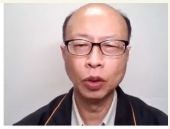




©2020 SCSE/NTU

22

CZ1106
CE1106

## Quiz: Instructions & Memory Access



- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.

a
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(1 MA)

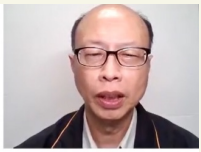
b
```
MOV R1,R0
LDR R1,[R0]
```
(2 MA)
(2 MA)

c
```
MOV R1,R0
LDR R1,[R0]
```
(2 MA)
(1 MA)

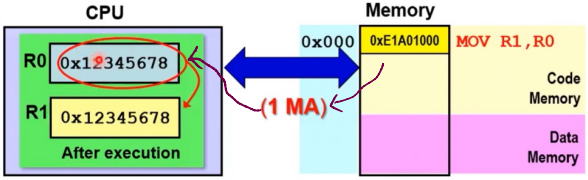
d
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(2 MA)

CZ1106
CE1106

## Quiz: Instructions & Memory Access



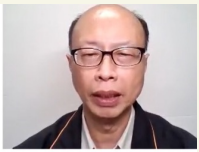
- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.



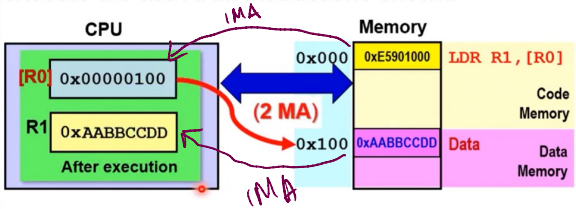

d
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(2 MA)

106
106

## Quiz: Instructions & Memory Access



- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.

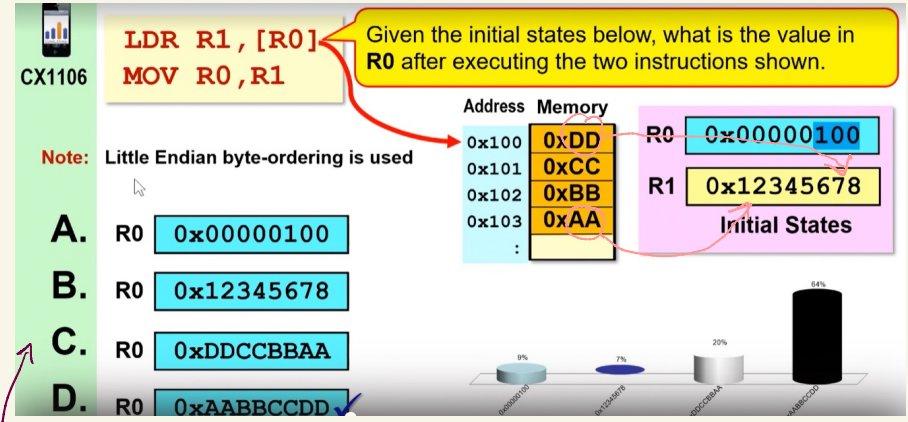



d
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(2 MA)

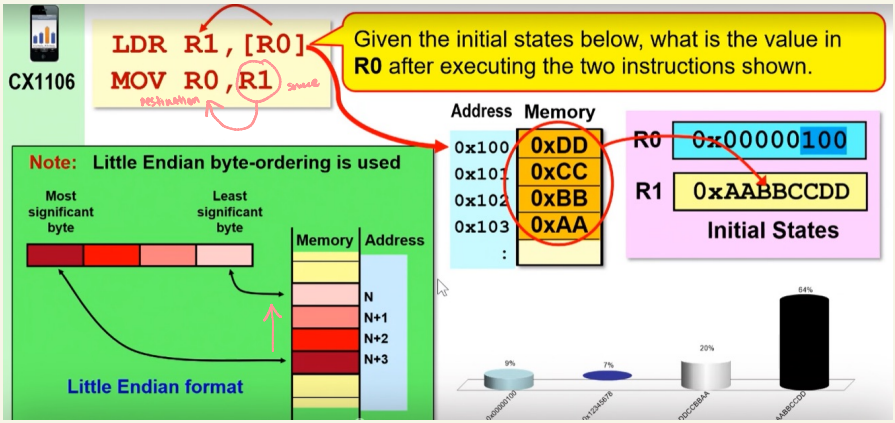
In the ARM instruction set, you cannot really specify the address of memory location directly, have to put into a register

then use register like a pointer (points to memory location)

Lecture Questions




big Endian

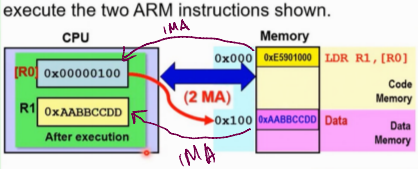



MSB on higher address, LSB on lower address

CZ1106
CE1106

## Quiz: Instructions & Memory Access

- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.




d. `MOV R1,R0` (1 MA)
`LDR R1,[R0]` (2 MA)

CZ1106
CE1106

CX1106

## Reducing Memory Cycles

```
LDR R1,[R0]  2~
LDR R2,[R0]  2~
LDR R3,[R0]  2~
             6
```

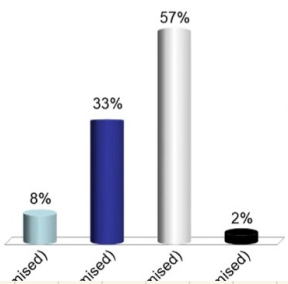
- What is the **number of memory cycles** needed to execute of this code segment?
- What is the minimum cycle count if this code is **fully optimised**?

A. 3 and 3(optimised)

B. 6 and 3(optimised)

C. ✓ 6 and 4(optimised)

D. 6 and 5(optimised)




```
① LDR R1,[R0]  2~
② MOV R2,R1    1~
③ MOV R3,R1    1~
Fully optimised = 4
```

① Access the memory and get it into a register

② Use result stone in R1 and replicate to R2

③ same as ②

CZ1106
CE1106

## Program Execution (Summary)

- Execution of a single instruction may involve multiple accesses to memory. In most single memory (von Neumann-type) CPU, different instructions take **different number** of clock **cycles** to execute.
- Data transfer on external bus is slower than within CPU's internal bus. μP system performance is limited by data traffic bandwidth between CPU and memory (also known as the **von Neumann bottleneck**).
- Keeping regularly used operands in CPU **registers** helps reduce memory access.
- Keeping instructions and data in separate memories (**Harvard architecture**) can help make instructions execute in more regular cycles (using parallel fetches) and thus improve performance.

| R0 |
|---|
| R1 |
| R2 |
| R3 |
| R4 |
| R5 |
| R6 |
| R7 |
| R8 |
| R9 |
| R10 |
| R11 |
| R12 |
| R13 (Stack Pointer) |
| R14 (Link Register) |
| R15 (Program Counter) |

23

→ do not have to be fetched; reduce memory access

Data is transferred between memory & CPU by the external data bus