IG CS Topic 3.1 - 3.5 Computer Architecture

Created by HardyWen

The Role of CPU

- **Von Neumann architecture**
  - the design of the current computer systems
- **Processors**
  - **Central Processing Unit (CPU)**
    - a component in a computer system that processes data and instructions
    - Key parts of the CPU
      - **Control Unit (CU)**
        - the component in the CPU that controls all the operations in the CPU
      - **Arithmetic logic unit (ALU)**
        - the component in the CPU that performs all the mathematical and logical operations required when processing data and instructions
  - **Microprocessor**
    - an **integrated circuit** in **a single chip** that is able to perform many of the functions of a CPU
- **Embedded system**
  - a computer system that performs a dedicated function
  - E.g. Traffic lights, washing machines, and digital alarm clock
  - **Features**
    - small size
    - configurability
    - fast and lightweight
  - **Characteristics**
    - needed minimal user interface
    - limited memory, low cost, fewer power consumptions
    - designed for one specific task

Fetch-decode-execute cycle

- the cycle through which data and instructions are processed
- **Glossary**
  - **Random access memory (RAM)**
    - a component in the CPU that holds data and programs that are currently in use
    - data is stored in RAM when the data and instructions are input

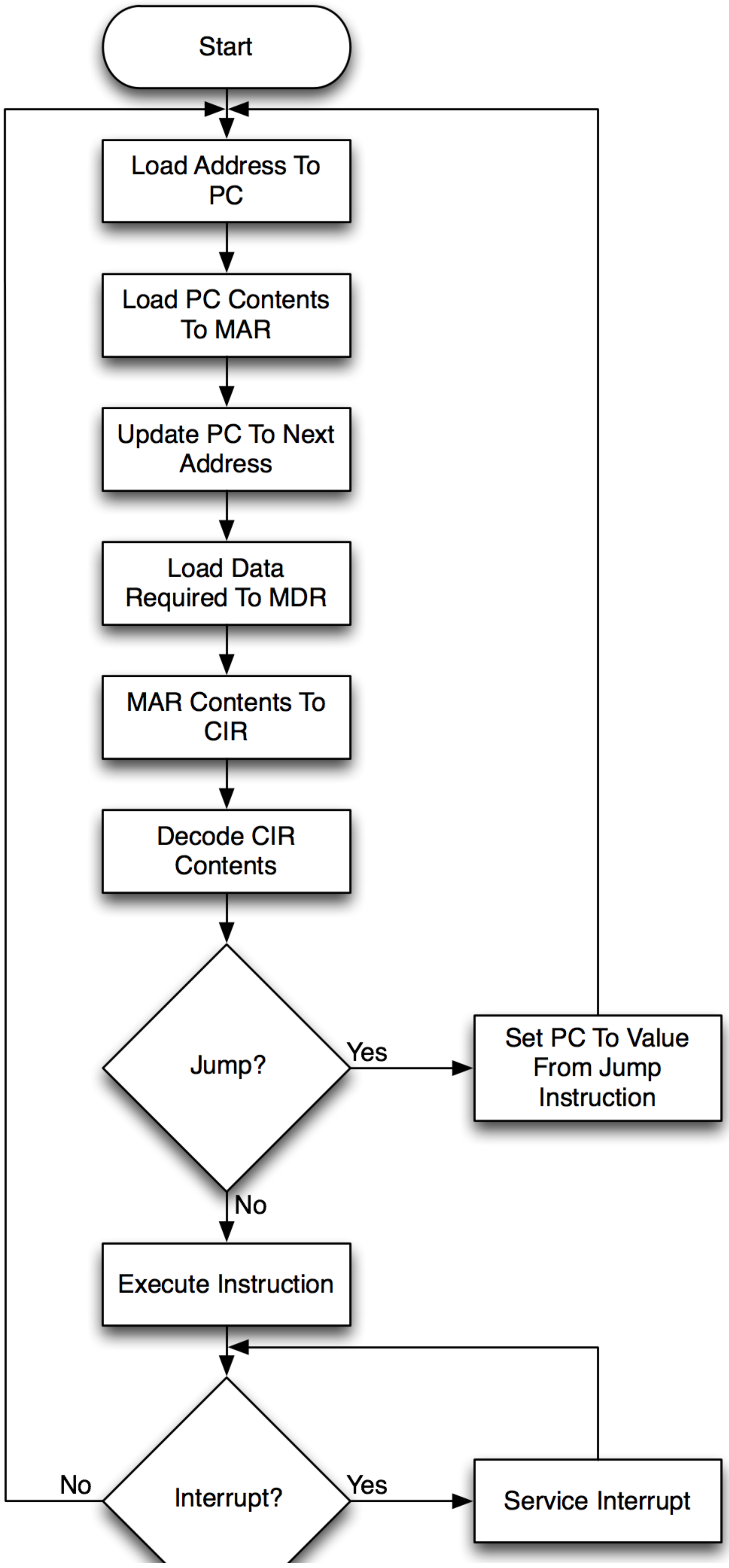
The concepts below are **registers**.

    - **Program counter (PC)**
        - a component in the CPU that stores the address of the next instruction to be processed
    - **Memory address register (MAR)**
        - a component in the CPU that holds the address of the data or instruction to be located in RAM
    - **Memory data register (MDR)**
        - a component in the CPU that holds the data or instructions that are fetched from RAM
    - **Current instruction register (CIR)**
        - a register that is built into the **CU** that holds the current instruction that is being processed in the CPU
    - **Accumulator (ACC)**
        - a register that is built into the ALU that stores the result of any interim calculations

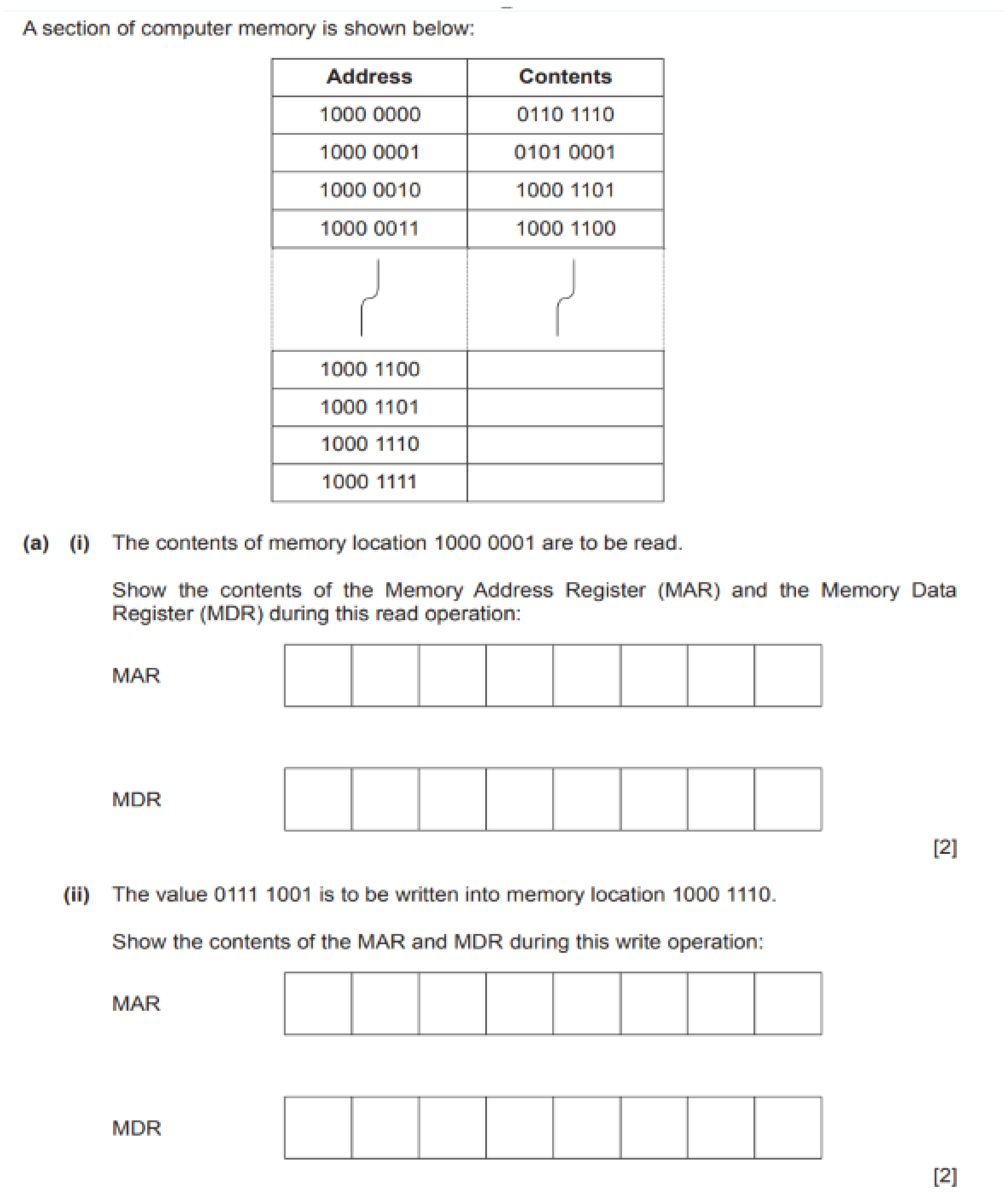
    The concepts below are **buses**.

    - **Data bus**
        - wires used for the transmission of **data and instructions** between components in a computer
    - **Control bus**
        - wires used for the transmission of **control signals** between components in a computer
        - the signals from CU are sent using the control bus to tell the components when they need to perform their different roles
    - **Address bus**
        - wires used for the transmission of **addresses** between components in a computer
- **Stages**
    - **The Fetch stage**
        - the instruction is fetched **from the RAM into the CPU**
        - the data or addresses or instructions travel through busses
        - The graph shows the connection between components in the fetch stage
    - **The Decode stage**
        - the instruction is decoded within the CPU
        - the instruction needs to be decoded so that the CPU can understand what is required to execute the instruction
        - the **CU** uses an **instruction set** to decode the instructions into readable commands that are in machine code

    - **The Execute stage**
        - actions that are required for the instruction are carried out
        - If calculations occur, such as the "add" command, the data would be transferred to the ALU to be calculated and the result would be stored in ACC
- **Process**


Start
Load Address To PC
Load PC Contents To MAR
Update PC To Next Address
Load Data Required To MDR
MAR Contents To CIR
Decode CIR Contents
Jump?
Yes
Set PC To Value From Jump Instruction
No
Execute Instruction
Interrupt?
No
Yes
Service Interrupt

Key Problem



A section of computer memory is shown below:

Address	Contents
1000 0000	0110 1110
1000 0001	0101 0001
1000 0010	1000 1101
1000 0011	1000 1100
⌇	⌇
1000 1100	
1000 1101	
1000 1110	
1000 1111	

(a) **(i)** The contents of memory location 1000 0001 are to be read.

Show the contents of the Memory Address Register (MAR) and the Memory Data Register (MDR) during this read operation:

MAR								
MDR								

[2]

(ii) The value 0111 1001 is to be written into memory location 1000 1110.

Show the contents of the MAR and MDR during this write operation:

MAR								
MDR								

[2]

CPU Performance

- **Number of cores**
  - **Core**
    - the part of the CPU that contains all the components that are used to perform the fetch-decode-execute cycle
  - The more the core, the more instructions can be processed each time
    - because a core more means that the computer can process one more FDE cycle and once

    - **Quad core**
        - four cores
    - **Dual core**
        - two cores
    - But 2 cores don't mean that it is 200% more efficient than a single core!
        - if your program can use all 4 processors, then the quad-core will then be about 70% quicker than the dual-core processor
- **The clock speed**
    - the number of FDE cycles that can be performed in a second
    - the higher the clock speed, the higher the performance
    - a CPU with a clock speed of **2 GHz (gigahertz)** can process **2 billion** instructions at once
- **The cache size**
    - **cache**
        - a type of storage that is **built into the CPU**, to store the most frequently used data and instructions
        - CPU might fetch the most recent and frequent data from the cache but not RAM
    - the larger the cache size, the higher the performance
        - larger cache means that more frequent or recent data can be stored, spending less time going to the RAM
    - larger cache **does not** always increase the performance
        - if the cache size is too large, its performance would decrease due to longer processing time

IG CS Topic 4.4-4.6 Programming Languages

created by Hardy Wen

Types of Programming language

- **Definitions**
    - **High-level language (Hll)**
        - uses human-language style words
            - e.g. if, while, output, print, input
        - examples include Python and C++
    - **Low-level language (Lll)**
        - included **machine code** and **assembly language**
        - **Machine code**
            - binary code, an example of a low-level language
            - **non portable**
                - cannot be run on different types and manufacturers of computers
        - **Assembly language**
            - code written in mnemonics that allows direct manipulation of the hardware
            - must be converted into binary code to run

        - using an **assembler**
- **Difference**
    - Remember to refer to the context when answering the questions
    -

High-level language	Low-level language
Easier to understand, read, and write	Harder to understand, read, write
Portable, machine independent	Not Portale, machine dependent
Explicit - one statement can represent many low-level instructions	Several instructions are needed for each high-level statement
Cannot directly manipulate the hardware	Can directly manipulate the hardware
Easier to debug	Harder to debug
Slower execution: translation process is slow	More efficient in speed and memory usage

Translators

- a type of software that converts encode written in one programming language into another, usually a high-level language into a low-level language
- **Assembler**
    - converts the assembly language into machine code
- **Compiler and Interpreter**
    - Both of these translators convert a high-level language into a low-level language
    -

Interpreters	Compilers
translates and executes line by line	translates and executes as a whole file
reports the syntax error one by one until it is fixed	reports the all the errors at once
codes need to be translated each time running, so needs the source code and the interpreter software to run	produces an executable file that can be run without the source code and the compiler
tests can occur without completing the whole code	have to finish a section of code before testing it

    - **Model Answer: Usage of *Interpreters***
        - Remember to refer to the context when answering the questions
        - An interpreter is useful when developing the code. This is because an interpreter runs the code line by line and stops until the current error has been fixed, which offers it the ability to show syntax error one by one and make people to comprehend the error and fix it easily. In addition, you can test a section of the

    whole code while developing using an interpreter, which also makes it easier to debug.
    - However, an interpreter might not be helpful when the code has already been developed and is ready for sell or distribution. This is because the source code and the interpreter software must exist in order to run the code, and the end users must wait for the program to be interpreted before run it.
  - **Model Answer: Usage of *Compilers***
    - Remember to refer to the context when answering the questions
    - A compiler is useful when the code is finished and is ready for sell. This is because a compiler creates an executable file which allows the program to be ruined without the compiler software and the source code. Therefore, the users would not need to download the software in advance to run the program, and the no-requirement for the source code protects the intellectual property and decreases the risk of being hacked as sometimes people can find out bugs through the source code and operate some malice attacks based on the bugs.
    - However, a compiler might not be helpful when developing the code. This is because compilers run the code as a whole, and hence would reveal all the errors at once, making it hard to debug as so much bugs might mess up.

Integrated Development Environment (IDE)

- a piece of software that allows a user to write, test and run program code
- **Parts of an IDE**
  - **Editor**
    - a feature of an IDE that allows the user to enter and amend program code
    - **Auto-completion**
      - a feature of an editor that gives the user options when they start typing a command
    - **Auto-correction**
      - a feature of an editor that identifies spelling error and changes them
    - **Prettyprint**
      - a feature of an editor that changes the color of the text such as highlight key words
    - Block minimising
      - a feature of an editor that minimizes a section of code
  - **Translator**
    - a feature of an IDE that provides a relevant translator for the programming language
  - **Run-time environment**
    - a feature of an IDE that allows a program to be run and lets the user interact with the program
    - features that help debugging
      - *Break points*
        - the user can set break points and the program will stop running on this line

    - to check outputs and values
- *Variable watch window*
    - the window which reveals the variables in the program and its current value
- *Stepping*
    - the program runs line by line, and the user needs to tell the program to move on
    - allows the program to be checked line by line