FIT1047 Week 4 Workshop

FIT1047 Introduction to Computer Systems, Networks and Security

Pre-Workshop Lecture Content

1. Memory
2. The memory hierarchy

Understand the layout of computer main memory

3. Indirect addressing

Understand indirect addressing

4. Arrays and strings
5. Subroutine

Be able to write simple assembly code programs that use indirect addressing and subroutines

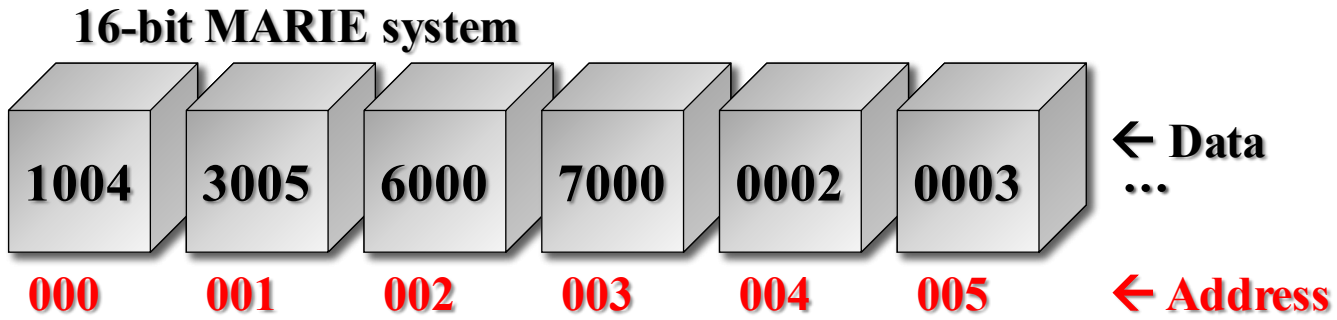
Learning Outcomes

At the end of this workshop, you will be able to:

1. Understand the layout of computer main memory
2. Understand indirect addressing
3. Be able to write simple assembly code programs that use indirect addressing and subroutines

Memory Organization

Think of it as a sequence of "boxes":



We give each box an **address**: the number of the box, starting from 0.

Each box contains a **value** (here: a 16-bit number).

This could be a **machine code instruction**, or **data**.

Programs can **read** and **change** the value stored at a location.

12 bits (000 to FFF)

Total addressable locations:

$2^{12} = 2^2 \times 2^{10} = 4K$

16 bits per location

Total memory size:

$4K \times 16 = 64$ Kbits

16-bit word-addressable system:

4K × 16 word-addressable memory module

Addressable locations

Volume of a location (bits)

System word size

Activity 1: Addressing Memory

Assuming that one memory chip is addressed as 2K x 8 byte-addressable, a memory contains multiple chips that builds a 64 x 2K x 8 byte-addressable memory module, answer the following questions?

a. How many chips required for 64 × 2K × 8 byte-addressable memory?

b. How many memory locations does this memory has?

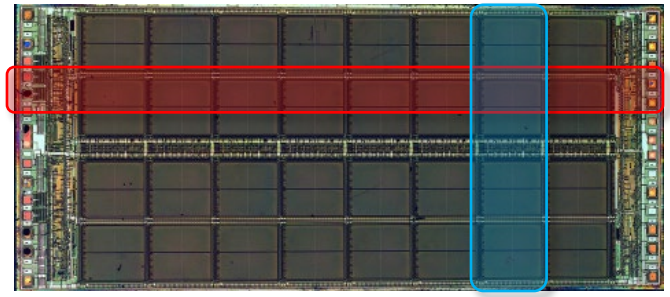
c. How many bits required to address one memory location?

d. What is the memory size?

Addressable locations	Volume of a location (bits)	System word size
64×2K	8	byte-addressable

a. 2K×8 per chips

No. of chips = (64×2K×8)/(2K×8)
= 64 chips



b. 64×2K locations

$2^6 \times 2^1 \times 2^{10} = 2^{17}$ locations

c. 2^{17} locations

No. of bits = $\lceil \log_2 2^{17} \rceil$
= 17 bits

010 3 bits for 8 rows

110 3 bits for 8 columns

00010010011 11 bits for 2K locations

d. 64×2K×8

Memory size = $2^{17} \times 8$
= $2^{17} \times 2^3$
= 2^{20} bits
= 1 Mbit

Quiz (30 minutes)

Opens: Monday, 11 November 2024, 8:00 AM
Closes: Monday, 11 November 2024, 11:55 PM

This in-class test will assess your knowledge of binary numbers, character encodings and Boolean logic gates.

You need to **attend the test in person**, during your **allocated workshop** in Week 4. Do not go to any other sessions! Remember to bring along your student ID card (physical card or the digital M-PASS) or any kind of photo ID for authentication purpose.

You will be provided with a password at the start of the workshop.

Do not share this password with other students. Any attempt of this test outside of the classroom will be reported as a breach of academic integrity.

Late comers will not be given additional time to complete the quiz. So please come on time!

Open Notes Instruction: You are allowed to bring and read any notes during the quiz (can be paper-based or digital PDF files). ASCII table will be provided to you. However, you are **NOT** allowed to use any calculator, any other software (e.g. Logisim, Excel, language translator), any service platform (e.g. Google, ChatGPT), any communication tool (e.g. email, SMS, Whatsapp, WeChat) during the quiz. You may also bring a few pieces of blank paper for you to write down the draft or calculation steps, though we will not collect or mark these papers. **All answers must be inputted in Moodle.**

- **Practice Quiz:** Link

Attempts allowed: 1

To attempt this quiz you need to know the quiz password

Time limit: 30 mins

Summary

ASSESSMENT	DUE
Assignment 1 Part 1a submission (Weight: 0%)	Friday, 8 November 2024, 11:55 PM
Assignment 1 Part 1b submission (Weight: 7.5%)	Friday, 8 November 2024, 11:55 PM
Assignment 1 Part 2 Quiz (Weight: 7.5%)	Monday, 11 November 2024, 11:55 PM

Get ready, make sure that:

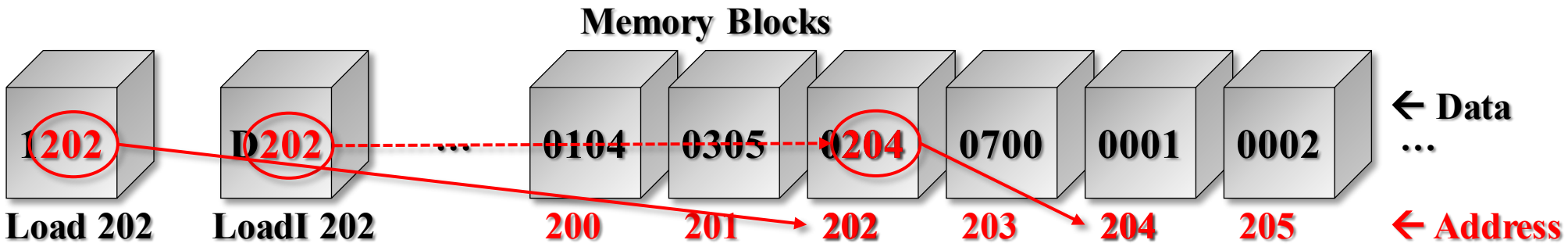
1. Device has sufficient power or charging is available.
2. Stable network connection.
3. No pending OS updates.
4. Logged into Moodle.

Learning Outcomes

At the end of this workshop, you will be able to:

1. Understand the layout of computer main memory
2. Understand indirect addressing
3. Be able to write simple assembly code programs that use indirect addressing and subroutines

MARIE – Indirect Addressing



Load 202 → Load value from address **202** into **AC**. → **AC = 0204**

LoadI 202 → Look up value stored at address **202**. → **M[202] = 0204**
Then load value from that address into **AC**. → **AC = 0001**

Load 202
AC ← M[202]
AC ← 0204

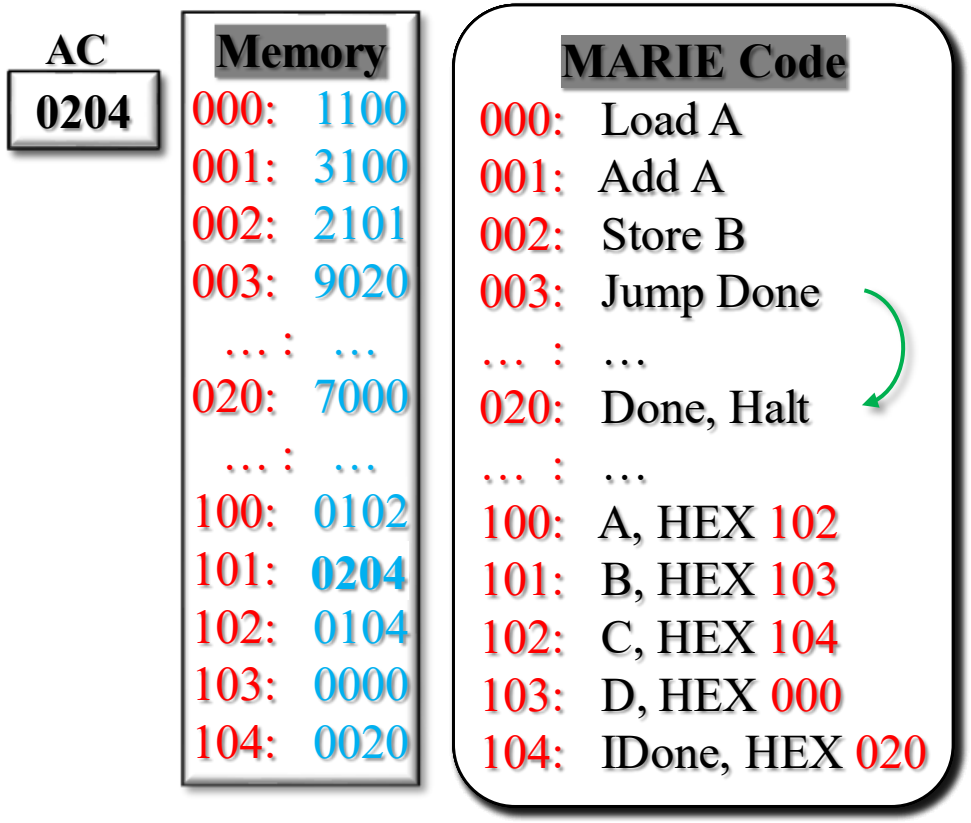
LoadI 202
AC ← M[M[202]]
AC ← M[204]
AC ← 0001

Decode
Decode opcode in IR[15-12]
5. MAR ← X or IR[11-0]
6. MBR ← M[MAR]

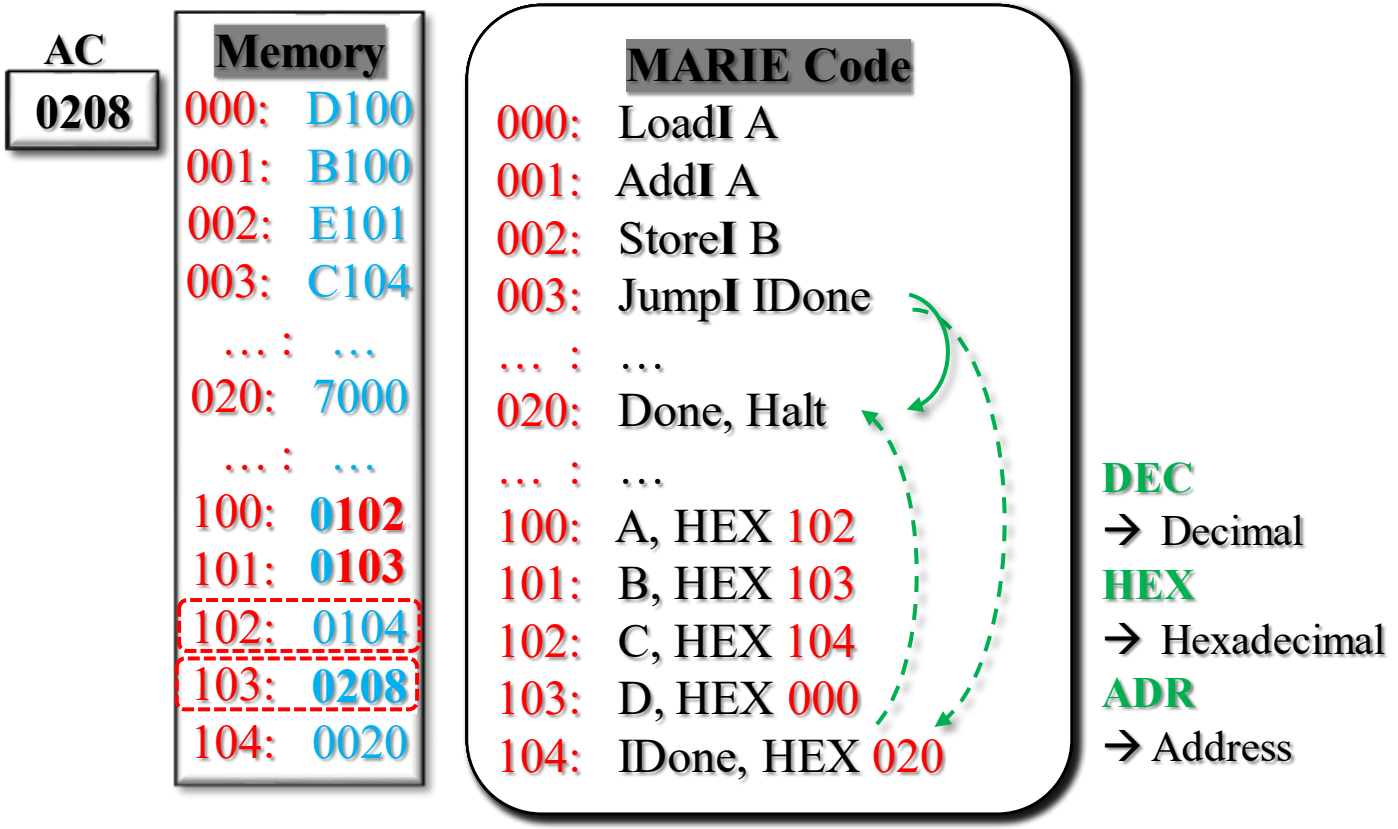
Instruction	Execute
AddI X	7. MAR ← MBR
	8. MBR ← M[MAR]
	9. AC ← AC + MBR
JumpI X	7. PC ← MBR
LoadI X	7. MAR ← MBR
	8. MBR ← M[MAR]
	9. AC ← MBR
StoreI X	7. MAR ← MBR
	8. MBR ← AC
	9. M[MAR] ← MBR

MARIE – Direct vs Indirect Addressing

Direct Addressing



Indirect Addressing



JumpI IDone → Jump M[IDone] → Jump Done
JumpI 104 → Jump M[104] → Jump 020

Activity 2: MARIE – Indirect Addressing

Using MARIE.js, write a program to output "FIT1047"

Pseudocode

1. Initialise Temp to starting address
2. If Temp pointing to address holding value '0', jump to Step 6
3. Output value from address pointing by Temp
4. Increase Temp to next address
5. Jump to Step 2
6. Halt

- Check when to use direct/indirect instructions
- Always terminate string with '0'

LoadI X	E	Load **Indirect**: Use the value at address **X** as the **actual address** to loads the contents into **AC**

MARIE Code

```
        Load Pointer
        Store Temp
Loop,   LoadI Temp
        Skipcond 400
        Jump Else
        Jump If
Else,   Output
        Load Temp
        Add One
        Store Temp
        Jump Loop
If,     Halt
```

-continue

```
One,    DEC 1

Temp,   HEX 0
Pointer, ADR String
String, HEX 46
        HEX 49
        HEX 54
        HEX 31
        HEX 30
        HEX 34
        HEX 37
        HEX 0
```

-continue

```
F,      HEX 46
I,      HEX 49
T,      HEX 54
One,    HEX 31
Zero,   HEX 30
Four,   HEX 34
Seven,  HEX 37
```

MARIE Code

```
Load F
Output
Load I
Output
Load T
Output
Load One
Output
Load Zero
Output
Load Four
Output
Load Seven
Output
Halt
```

Try output 2nd string from sample code

Sample code in Notes below ↓

Activity 2: MARIE – Indirect Addressing

Using MARIE.js, write a program to add all the values hardcoded in the memory locations. The values are 1, 2, 4, 8, 16, 32, 64, 128. Output the value and store at the address HEX 030.

Pseudocode

**Pointer* points to starting address
**Ans* starts with '0'
**Result* points at HEX 30

1. Ans = Ans + M[Pointer]
2. Pointer = Pointer + 1
3. If Pointer points to address holding value '0', jump to Step 5
4. Else, Jump to Step 1
5. Store Ans to address pointed by Result
6. Output and Halt

MARIE Code

```
Loop, Load  Ans
      AddI  Pointer
      Store Ans
      Load  Pointer
      Add   One
      Store Pointer
      LoadI Pointer
      Skipcond 400
      Jump  Loop
      Load  Ans
      StoreI Result
      Output
      Halt
```

-continue

```
One,     DEC 1
Ans,     DEC 0
Result,  HEX 30

Pointer, ADR Bin
Bin,     DEC 128
         DEC 64
         DEC 32
         DEC 16
         DEC 8
         DEC 4
         DEC 2
         DEC 1
         DEC 0
```

AddI X	B	Add **Indirect**: Use the value at Address **X** as the **actual address** of the data operand to add to **AC**
StoreI X	D	Store **Indirect**: Use the value at Address **X** as the **actual address** to stores the contents of **AC**
LoadI X	E	Load **Indirect**: Use the value at address **X** as the **actual address** to loads the contents into **AC**

Try modifying this program:
Convert an 8-bit binary input, entering one bit at a time, into its decimal equivalent. Post your code to **Ed Forum**.

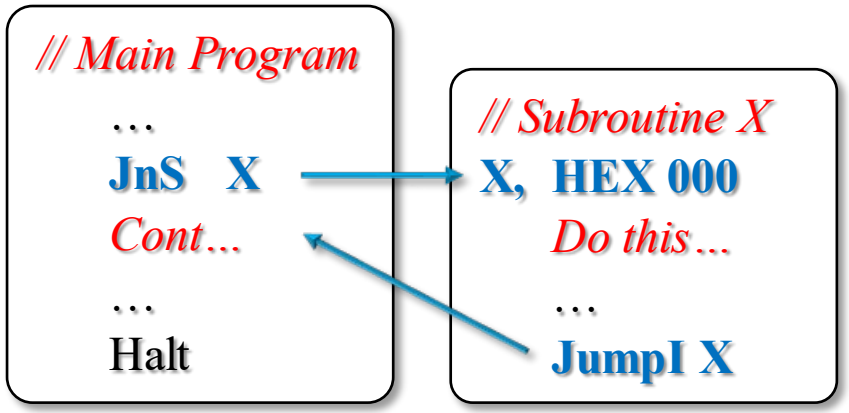
Sample code in Notes below ↓

Learning Outcomes

At the end of this workshop, you will be able to:

1. Understand the layout of computer main memory
2. Understand indirect addressing
3. Be able to write simple assembly code programs that use indirect addressing and subroutines

MARIE – Subroutine



Remember:

1. Call subroutine with **JnS**
2. Exit subroutine with **JumpI**
3. Define subroutine label as **HEX 000**
4. **Jump** is allowed within a routine, but NOT between different routines

Subroutine

- a section of code that performs a **specific** task

JnS X → Call/execute subroutine **X**

a) Jump and Store PC at memory location **X**

b) PC = X + 1

JumpI X → Exit and return from subroutine **X**

Jump to address (return-PC) stored at **X**

JumpI X → Jump M[X] → Jump return-PC

MARIE – Subroutine

Compute Z = 2(X-Y).

i) Convert the programs to subroutines

ii) Prompt user inputs X and Y, then execute Z = 2(X-Y) using subroutines in main program.
→ step 1: Z = X-Y
→ step 2: Z = Z+Z

Remember:

1. Call subroutine with **JnS**
2. Exit subroutine with **JumpI**
3. Define subroutine label as **HEX 000**
4. **Jump** is allowed within a routine, but NOT between different routines

```
// Main Program
/ Inputs to X, Y
/ ...
/ Pass (X,Y) to (SubX,SubY)
/ and call for SubSubt
      Load X
      Store SubX
      Load Y
      Store SubY
      JnS SubSubt
      Load SubZ
      Store Z
/ Pass (Z,Z) to (AddX,AddY)
/ and call for SubAdd
/ ...
      Load Z
      Output
      Halt
X,    DEC 0
Y,    DEC 0
Z,    DEC 0
```

```
// Program for
// AddX+AddY=AddZ

      Load  AddX
      Add   AddY
      Store AddZ
      Halt
AddX,     DEC 0
AddY,     DEC 0
AddZ,     DEC 0
```

```
// Program for
// SubX-SubY=SubZ

      Load  SubX
      Subt  SubY
      Store SubZ
      Halt
SubX,     DEC 0
SubY,     DEC 0
SubZ,     DEC 0
```

Sample code in Notes below ↓

Activity 3: MARIE – Subroutine

Using MARIE.js, write a program that allows user to enter three inputs, the first two are decimal values (X and Y) while the third input is a character either "a" or "s" for addition (X + Y) and subtraction (X – Y) operations. Use subroutines to create the addition and subtraction operations.

MARIE Code

```
// Prompt for 3 inputs
Input
Store X
Input
Store Y
Input
Store Choice
```

-continue

```
/ Check character 'a'
CheckA, Load  Choice
        Subt  CharA
        Skipcond 400
        Jump  CheckS
        JnS   subAdd
        Jump  Done
/ Check character 's'
CheckS, Load  Choice
        Subt  CharS
        Skipcond 400
        Jump  Invalid
        JnS   subSubt
        Jump  Done
```

-continue

```
/ Invalid Choice
Invalid,  Clear
          Store Z
Done, Load Z
      Output
      Halt
// Define variables
X,    DEC 0
Y,    DEC 0
Z,    DEC 0
Choice,  DEC 0
CharA,   HEX 61
CharS,   HEX 73
```

-subroutines

```
/ Subroutine for addition
subAdd, HEX 000
        Load  X
        Add   Y
        Store Z
        JumpI subAdd
/ Subroutine for subtraction
subSubt,  HEX 000
          Load  X
          Subt  Y
          Store Z
          JumpI subSubt
```

JnS X	0	Jumps and Store: Stores value of PC at Address **X** then increments PC to **X+1**
JumpI X	C	Jump **Indirect**: Use the value at Address **X** as the **actual address** to jump to

Sample code in Notes below ↓

MONASH University

Post-class Activity

1. **Access ChatGPT:** Go to chatgpt.com and sign up using your Monash account. The Free tier should be sufficient for this activity.
2. **Answer the Following Questions Using ChatGPT:**
   - What is the function of a subroutine in programming?
   - What are the key properties of a well-designed subroutine?
   - What is the function of comments and documentation in a program?
   - What makes comments or documentation good and useful in a program?
   - What are suitable naming conventions for labels, variables, and subroutines?
3. **Using ChatGPT:** Copy and paste each question into ChatGPT's input area. Feel free to ask ChatGPT to refine or expand on the answers, or to generate alternative responses to deepen your understanding.
4. **Summarize and Share:** Summarize the key points from ChatGPT's responses and post your findings on **Ed Forum**.

MONASH University