

# FIT1047 – Week 4

## Memory and Indirect Addressing

# Overview

- Memory organisation
  - how to address different locations
- Instructions for accessing memory
  - indirect addressing
  - subroutines

# Memory

For the programmer:

- A sequence of locations
- Each location has an address
  - an unsigned integer, starting from 0
- Each location stores one value
  - each value has a fixed width (number of bits)
- We can read and change the value stored at a location

# Memory

Address	Hex value
000	1004
001	3005
002	2006
003	7000
004	008E
005	0D80
006	0000

(all values given in hexadecimal)

# What does it store?

Address	Hex Value	Integer
000	1004	4100
001	3005	12293
002	2006	8198
003	7000	28672
004	008E	142
005	0D80	3456
006	0000	0

# What does it store?

Address	Hex Value	Integer	Bit pattern
000	1004	4100	00010000000010100
001	3005	12293	00110000000000101
002	2006	8198	00100000000000110
003	7000	28672	01110000000000000
004	008E	142	0000000010001110
005	0D80	3456	0000110110000000
006	0000	0	00000000000000000

# What does it store?

Address	Hex Value	Integer	Bit pattern	Instruction
000	1004	4100	00010000000010100	Load 004
001	3005	12293	00110000000000101	Add 005
002	2006	8198	00100000000000110	Store 006
003	7000	28672	01110000000000000	Halt
004	008E	142	0000000010001110	JnS 08E
005	0D80	3456	0000110110000000	JnS D80
006	0000	0	00000000000000000	JnS 000

# What does it store?

Address	Hex Value	Integer	Bit pattern	Instruction
000	1004	4100	00010000000010100	Load 004
001	3005	12293	00110000000000101	Add 005
002	2006	8198	00100000000000110	Store 006
003	7000	28672	01110000000000000	Halt
004	008E	142	0000000010001110	JnS 08E
005	0D80	3456	0000110110000000	JnS D80
006	0000	0	00000000000000000	JnS 000

It stores bits!



# What does it store?

Address	Hex Value	Integer	Bit pattern	Instruction
000	1004	4100	00010000000010100	Load 004
001	3005	12293	00110000000000101	Add 005
002	2006	8198	00100000000000110	Store 006
003	7000	28672	01110000000000000	Halt
004	008E	142	0000000010001110	JnS 08E
005	0D80	3456	0000110110000000	JnS D80
006	0000	0	00000000000000000	JnS 000

It stores bits!

What they mean depends on how we interpret them!

# Addressing

In most architectures,

# Addressing

In most architectures,  
one memory location stores one byte.

# Addressing

In most architectures,  
one memory location stores one byte.  
Consequently, each location, each byte, has its own  
address.

# Addressing

In most architectures,  
one memory location stores one byte.

Consequently, each location, each byte, has its own address.

This is called byte-addressable.

# Addressing in MARIE

In MARIE,

# Addressing in MARIE

In MARIE,

one memory location stores one word (two bytes, or 16 bits).

# Addressing in MARIE

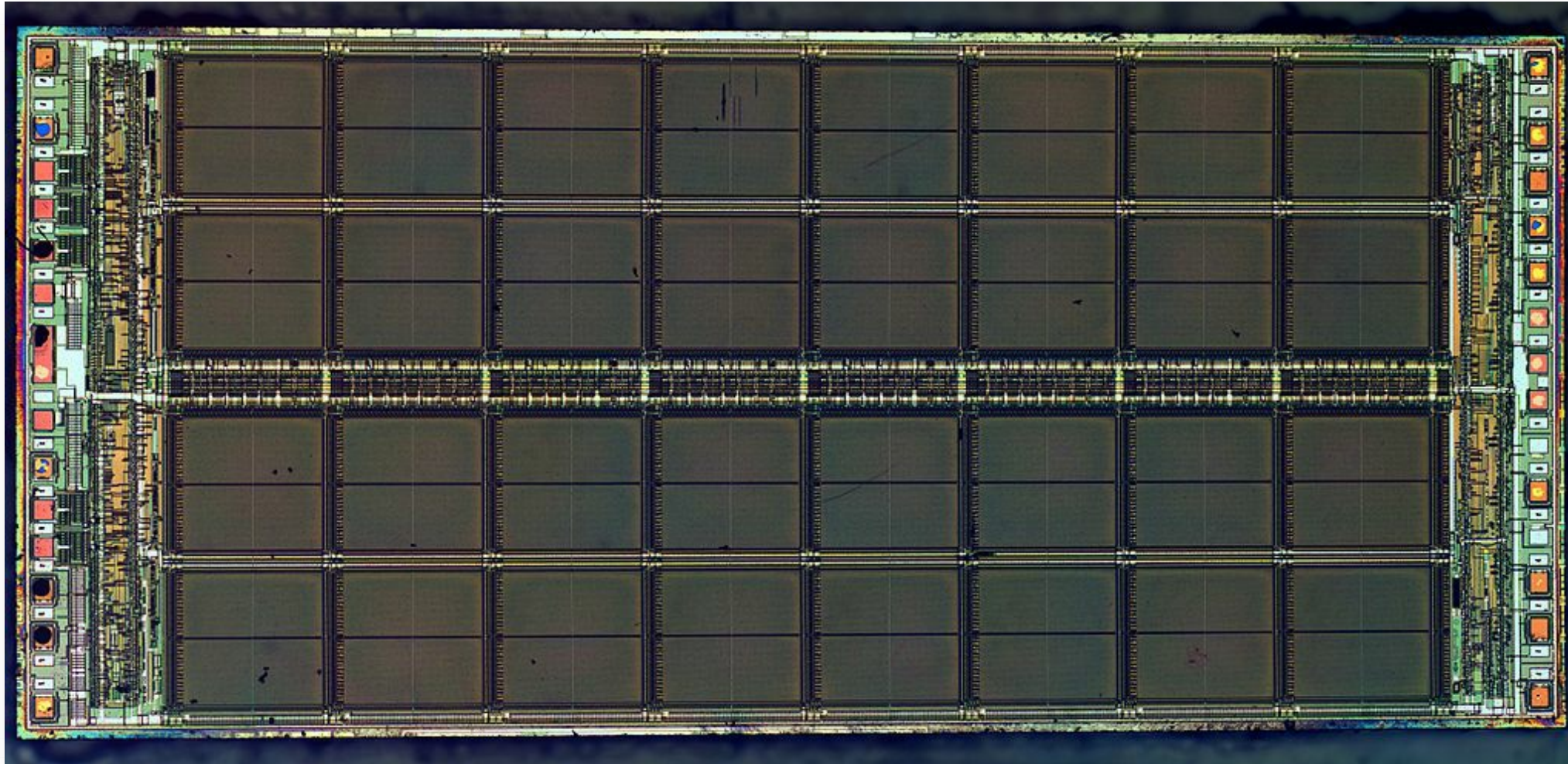
In MARIE,

one memory location stores one word (two bytes, or 16 bits).

An address therefore references a whole word.



# RAM



A DRAM chip with 1 megabit capacity. Source: Wikipedia

# Memory Organisation

RAM is made up of multiple chips.

Each has a fixed size  $L \times W$

- $L$ : number of locations
- $W$ : number of bits per location

E.g.  $2K \times 8$  means  $2 \times 2^{10}$  locations of 8 bits each.

# Memory Organisation

RAM chips are combined in rows and columns.

E.g. to build  $32K \times 16$  memory out of  $2K \times 8$  chips:

$2K \times 8$	$2K \times 8$
$2K \times 8$	$2K \times 8$
...	
$2K \times 8$	$2K \times 8$

How do we address individual locations?

# Memory Organisation

$2K \times 8$	$2K \times 8$
$2K \times 8$	$2K \times 8$
...	
$2K \times 8$	$2K \times 8$

$32K = 2^{15}$  words, so addresses need 15 bits.

# Memory Organisation

$2K \times 8$	$2K \times 8$
$2K \times 8$	$2K \times 8$
...	
$2K \times 8$	$2K \times 8$

$32K = 2^{15}$  words, so addresses need 15 bits.

- Use highest 4 bits to select the row

# Memory Organisation

$2K \times 8$	$2K \times 8$
$2K \times 8$	$2K \times 8$
...	
$2K \times 8$	$2K \times 8$

$32K = 2^{15}$  words, so addresses need 15 bits.

- Use highest 4 bits to select the row
- Use low 11 bits to select the word in the row

# Indirect Addressing

# Accessing memory in MARIE

So far:

- Store X
- Load X
- Add X
- Jump X



# Accessing memory in MARIE

So far:

- Store X
- Load X
- Add X
- Jump X

All use the value stored at address X.

# Accessing memory in MARIE

So far:

- Store X
- Load X
- Add X
- Jump X

All use the value stored at address X.

This is not very flexible!

# Indirect addressing

Add X:

Load the value stored at X and add it to the AC register.

AddI X:

Use the address stored at X, load the value from that address, and add it to AC.

("add indirect")

# Indirect addressing

Jump X:

Jump to address X.

Jumpl X:

Use the address stored at X, and jump to that address.

("jump indirect")

# Indirect addressing

## Advantages:

- addresses don't need to be hardcoded
- we can compute the address!
- e.g. loop through a list of values

# Adding up a list

000	Loop,	LoadI Addr	00C	One,	DEC 1
001		SkipCond 800	00D	Sum,	DEC 0
002		Jump End	00E	Addr,	HEX 00F
003		Add Sum	00F		DEC 70
004		Store Sum	010		DEC 73
005		Load Addr	011		DEC 84
006		Add One	012		DEC 0
007		Store Addr			
008		Jump Loop			
009	End,	Load Sum			
00A		Output			
00B		Halt			

# Subroutines

AKA procedures, functions, methods

A piece of code that

- has a well-defined function
- needs to be executed often
- we can call, passing arguments to it
- returns to where it was called from, possibly with a return value

# Subroutines

ISAs provide support for subroutines.

In MARIE,

JnS X:

Stores the PC into X, then jumps to X+1

X holds the return address

Jumpl X:

Jump to address stored at X

(returns to the calling code)



# Subroutines

MARIE example subroutine:

```

        Load  FortyTwo
        Store  Print_Arg
        JnS    Print
        Halt

FortyTwo, DEC 42

        / Subroutine that prints one number
Print_Arg, DEC 0          / put argument here
Print,   HEX 0           / return address
        Load  Print_Arg
        Output
        JumpI  Print      / return to caller
```

# Tutorials this week

- Programming MARIE
- Circuits for adding and subtracting

# Next lecture

## Wednesday

- Interrupts
- Input/Output

