CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

A Simple Computer

—Dr H. Asad

Overview

- Lecture 1 presented a general overview of computer systems.
- In Lecture 2, we discussed binary data arithmetic and how to perform additions and subtractions using complement's arithmetic.
- Lecture 3 described the fundamental components of digital gates and how to use these to build digital circuits.

- Having this background, we can now understand how computer components work, and how they fit together to create useful computer systems.

- **This lecture:**
  - Simple computer (revisited)
  - Instruction set architecture
  - MARIE
  - Programming at a low level
  - CISC & RISC architectures

Question

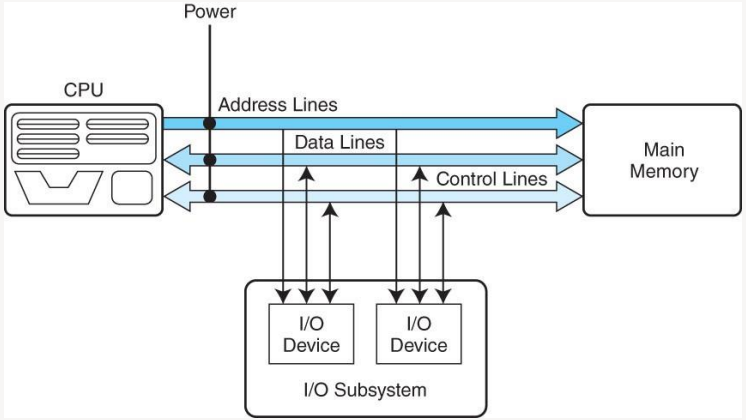
- How a code written in C/Python/Java gets executed on the hardware? For example the following C statement:

```
If  x > y
    x=x-1;
else
    y=y-x;
```

Overview of a Simple Computer

- Three main components:
  1) central processing unit (CPU)
  2) main memory
  3) Input/output (I/O) Subsystem

- The main operations of CPU:
  - processes instructions on data
  - reads from the Input and Writes to the Output
  - reads from and stores to memory
  - keeps track of what to do next



CPU Basics

- The two principal parts of the CPU are the

Datapath and the ***Control unit*****.**

1) The datapath consists of an arithmetic-logic unit (**ALU**) and storage units (**registers**)
   - The **arithmetic-logic unit (ALU)** carries out logical and arithmetic operations as directed by the control unit.
   - **Registers** hold data that can be readily accessed by the CPU. They can be implemented using D flip-flops. A 32-bit register requires 32 D flip-flops.
   - **ALU** and **registers** are **interconnected by a data bus** that is also **connected to main memory**.
2) Various CPU components perform **sequenced operations** according to **signals provided by its control unit**. The **control unit** determines which actions to carry out according to the values in a **program counter** register.

Clocks

- Every computer contains at least one clock that synchronises the activities of its components
- A fixed number of clock cycles are required to carry out each data movement or computational operation
- The clock frequency, measured in megahertz or gigahertz, determines the speed with which all operations are carried out
- Clock cycle time is the reciprocal of clock frequency
  - For example, an 800 MHz clock has a cycle time of 1.25 ns.

Clocks vs CPU Time

- Clock speed should not be confused with CPU performance
- The CPU time required to run a program is given by the general performance equation:

$$\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

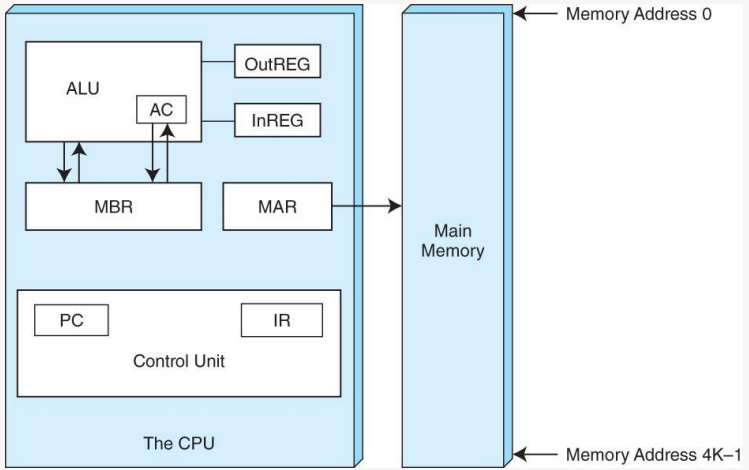
- CPU throughput can be improved by reducing
  - the number of instructions in a program
  - the number of cycles per instruction
  - the number of nanoseconds per clock cycle

MARIE, a simple architecture

- **M**achine **A**rchitecture that is:

  Really **I**ntuitive and **E**asy

- Why MARIE?
  - It is simple
  - You can download an emulator and play with it if you want



MARIE Register Space

AC: The Accumulator
- General purpose register holds data the CPU works with

MAR: Memory Address Register
- Holds the address of the data being referenced from/to memory

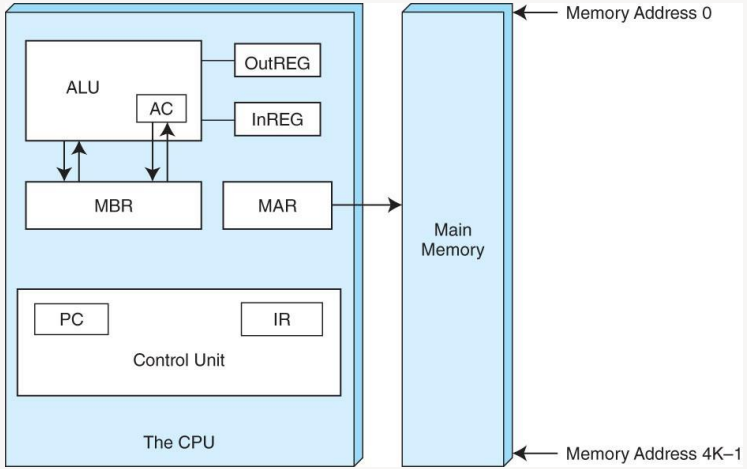
MBR: Memory Buffer Register
- Data read from or to be written to Memory

PC: Program Counter
- Address of the next instruction to be executed

IR: Instruction Register
- The next instruction to be executed

InREG: Input Register
- Holds the data received from the Input Device

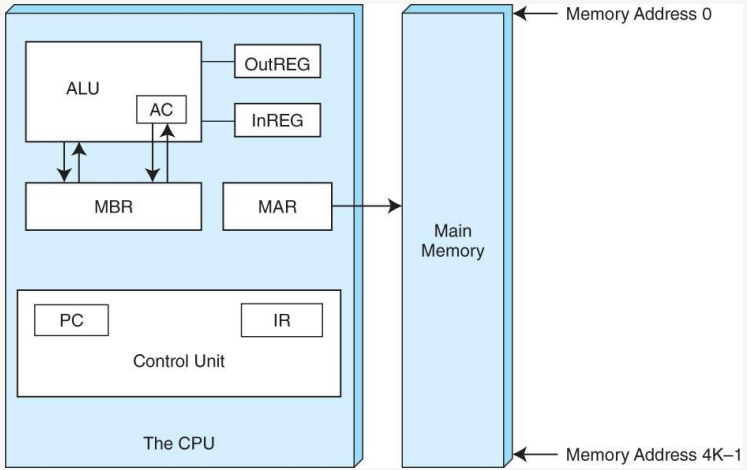
OutREG: Output Register
- Holds the data to be sent to the Output Device



MARIE’s Characteristics

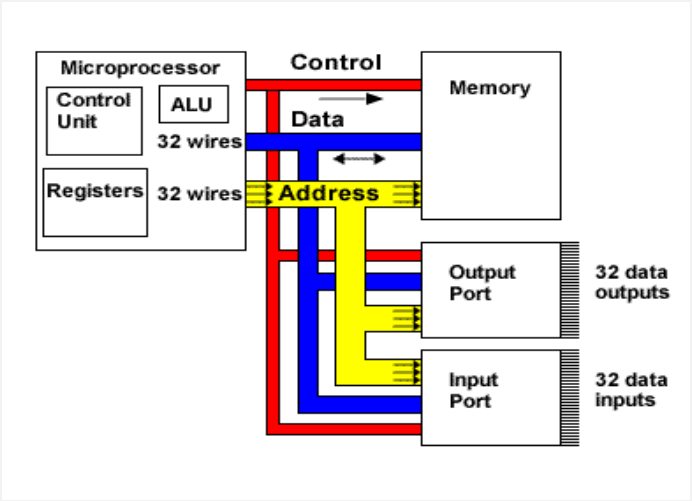
- 2’s complement binary data representation
- Stored program, fixed word length
- 4K words of word addressable main-memory
- 16-bit words for data
- 16-bit instructions

- **Registers**
- 16-bit accumulator (AC)
- 16-bit instruction register (IR)
- 16-bit memory buffer (MBR)
- 12-bit program counter (PC)
- 12-bit memory address register (MAR)
- 8-bit input register (InREG)
- 8-bit output register (OutREG)



Buses

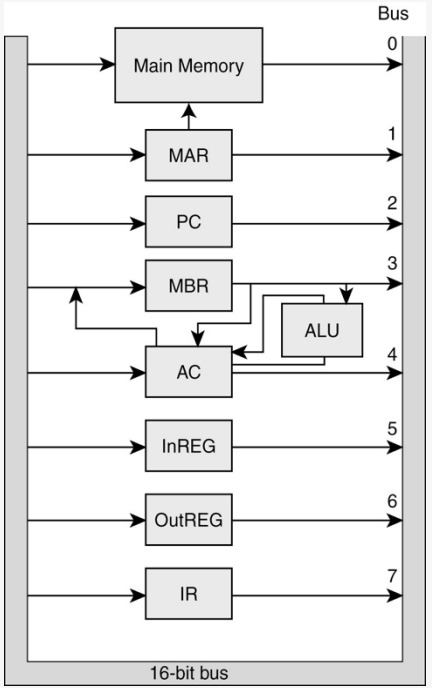
- A bus is a set of wires used to connect multiple subsystems within the computer. At any time, only one component (e.g., register, ALU, memory) can use the bus.



http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture1/lecture1-3-1.html

Datapath in MARIE

- MARIE has a 16 bit datapath which is a network of registers and arithmetic and logic units connected by bus



CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

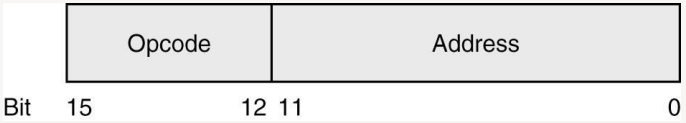
A Simple Computer: Instruction Set Architecture

—Dr H. Asad

Instruction Set Architecture (ISA)

- The **Instruction Set Architecture** of a computer specifies the instructions that the computer can perform
- It is the principal interface between the software and the hardware
- Some ISAs include hundreds of different instructions for processing data and controlling program execution, but **MARIE's ISA includes only 13 instructions**!

- ISA describes the format of the instructions, giving:
  the operation/type of the instruction (**opcode**) and
  the **address** of the operand



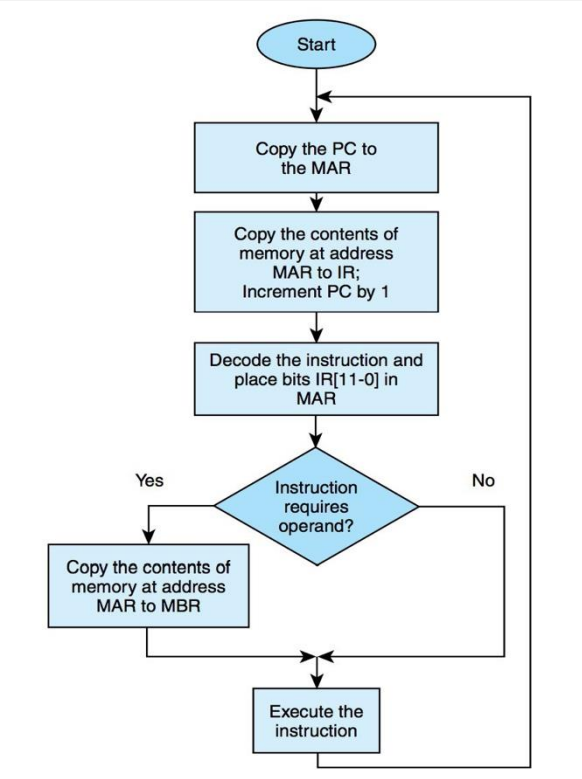
MARIE Fundamental Instruction Set

Instruction Number		Instruction	Meaning
Bin	Hex		
0001	1	Load *X*	Load the contents of address *X* into AC.
0010	2	Store *X*	Store the contents of AC at address *X*.
0011	3	Add *X*	Add the contents of address *X* to AC and store the result in AC.
0100	4	Subt *X*	Subtract the contents of address *X* from AC and store the result in AC.
0101	5	Input	Input a value from the keyboard into AC.
0110	6	Output	Output the value in AC to the display.
0111	7	Halt	Terminate the program.
1000	8	Skipcond	Skip the next instruction on condition.
1001	9	Jump *X*	Load the value of *X* into PC.

Instruction Processing – FDE Cycle

The **fetch-decode-execute (FDE) cycle** is the series of steps that a computer carries out when it runs a program.

1) We first **fetch** an instruction from memory, and place it into the IR.
2) Once in the IR, it is **decoded** to determine what needs to be done next
3) If a memory value (operand) is involved in the operation, it is retrieved and placed into the MBR.
4) With everything in place, the instruction is **executed**
   - PC register holds the address of the next instruction to be executed
   - PC ← PC +1 goes to the next instruction

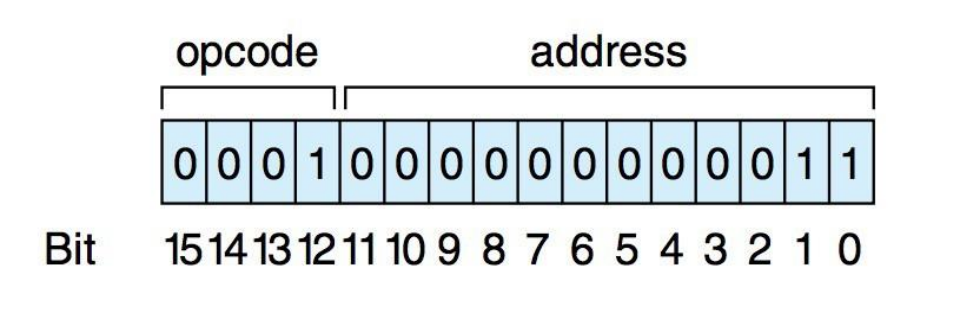


Instruction example: LOAD instruction

- From the human perspective

  `LOAD` 03_h

- This is a bit pattern (representation) of the above **LOAD** instruction as it would appear in the IR:



- We see that the opcode is 1 and the address from which to load the data is 03_h
- Both representations say *"load the data from address 03_h into the accumulator register AC"*

Microoperations

- **Each** (**ISA** level) **instruction** actually consists of a sequence of smaller instructions called **microoperations**
- The exact sequence of microoperations that are carried out by an instruction can be specified using the **register transfer language (RTL)** or **register transfer notation (RTN)**
- In the MARIE RTL, we use the notation **M[X]** to indicate the **actual data** value **stored in memory location X**, and ← to indicate the **transfer of bytes** to a register or memory location

Example Microoperations

- The RTL for the **LOAD X** instruction is:

```
MAR ¬   X
MBR ¬   M[MAR]
AC ¬   MBR
```

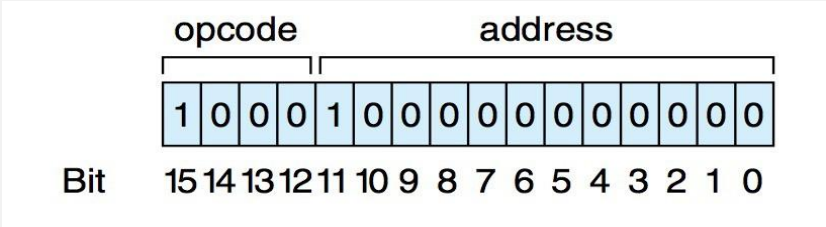
- Similarly, the RTL for the **ADD X** instruction is:

```
MAR ¬   X
MBR ¬   M[MAR]
AC ¬   AC + MBR
```

- Note that we do not have to name AC as the destination register in the original command. This register is implicit in the instructions

SKIPCOND Instruction

- SKIPCOND: skip the next instruction on condition
- Conditional branch, just like “if”
- This is a bit pattern for a **SKIPCOND** instruction as it would appear in the IR:



- The opcode is 1000_2
- **Bits 11 and 10 specify the condition** to be tested.
- The address part of the instruction is used to specify the condition

Microoperations of SKIPCOND

- Recall that **SKIPCOND** skips the next instruction according to the value of the AC
- The RTL for this instruction is the most complex in our instruction set:

```
If IR[11 - 10] = 00 then                                  (if bit 11 = 0 and bit 10 = 0)
            If AC < 0 then PC ← PC + 1

else If IR[11 - 10] = 01       then                       (if bit 11 = 0 and bit 10 = 1)

             If AC = 0 then PC ← PC + 1

else If IR[11 - 10] = 10       then                       (if bit 11 = 1 and bit 10 = 0)

             If AC > 0 then PC ← PC + 1
```

- Note: The notation **IR[11-10]** means the values of the 11th and 10th bit in register IR

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

A Simple Computer: MARIE FDE Cycle

—Dr H. Asad

A Simple Program

- Consider the simple MARIE program given below. We show a set of mnemonic instructions stored at addresses 0x100 – 0x106 (hex):

Hex Address	Instruction	Binary Contents of Memory Address	Hex Contents of Memory
100	Load 104	0001000100000100	1104
101	Add 105	0011000100000101	3105
102	Store 106	0010000100000110	2106
103	Halt	0111000000000000	7000
104	0023	0000000000100011	0023
105	FFE9	1111111111101001	FFE9
106	0000	0000000000000000	0000

A Simple Program (cont.)

In plain English

1. Load the contents of **M[104]** into **AC**
2. Add the contents of **M[105]** to **AC** and store result into **AC**
3. Store the value of **AC** into **M[106]**
4. Stop the program

Hex Address	Instruction	Binary Contents of Memory Address	Hex Contents of Memory
100	Load 104	0001000100000100	1104
101	Add 105	0011000100000101	3105
102	Store 106	0010000100000110	2106
103	Halt	0111000000000000	7000
104	0023	0000000000100011	0023
105	FFE9	1111111111101001	FFE9
106	0000	0000000000000000	0000

What does the computer do in the FDE cycle of this simple program?

- Remember the fetch-decode-execute (FDE) cycle
- For the **first instruction** (LOAD 104) at address 100

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	----	----	----	----
Fetch	MAR ← PC	100	----	100	----	----
	IR ← M[MAR]	100	1104	100	----	----
	PC ← PC + 1	101	1104	100	----	----
Decode	MAR ← IR[11-0]	101	1104	104	----	----
	(Decode IR[15-12])	101	1104	104	----	----
Get operand	MBR ← M[MAR]	101	1104	104	0023	----
Execute	AC ← MBR	101	1104	104	0023	0023

What does the computer do in the FDE cycle of this simple program?

- Remember the fetch-decode-execute (FDE) cycle
- For the **first instruction** (LOAD 104) at address 100

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	----	----	----	----
Fetch	MAR ← PC	100	----	100	----	----
	IR ← M[MAR]	100	1104	100	----	----
	PC ← PC + 1	101	1104	100	----	----
Decode	MAR ← IR[11-0]	101	1104	104	----	----
	(Decode IR[15-12])	101	1104	104	----	----
Get operand	MBR ← M[MAR]	101	1104	104	0023	----
Execute	AC ← MBR	101	1104	104	0023	0023

Inst: LOAD 104

What does the computer do in the FDE cycle of this simple program?

- Remember the fetch-decode-execute (FDE) cycle
- For the **first instruction** (LOAD 104) at address 100

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	----	----	----	----
Fetch	MAR ← PC	100	----	100	----	----
	IR ← M[MAR]	100	1104	100	----	----
	PC ← PC + 1	101	1104	100	----	----
Decode	MAR ← IR[11-0]	101	1104	104	----	----
	(Decode IR[15-12])	101	1104	104	----	----
Get operand	MBR ← M[MAR]	101	1104	104	0023	----
Execute	AC ← MBR	101	1104	104	0023	0023

Inst: LOAD 104

What does the computer do in the FDE cycle of this simple program?

- Remember the fetch-decode-execute (FDE) cycle
- For the **first instruction** (LOAD 104) at address 100

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	----	----	----	----
Fetch	MAR ← PC	100	----	100	----	----
	IR ← M[MAR]	100	1104	100	----	----
	PC ← PC + 1	101	1104	100	----	----
Decode	MAR ← IR[11-0]	101	1104	104	----	----
	(Decode IR[15-12])	101	1104	104	----	----
Get operand	MBR ← M[MAR]	101	1104	104	0023	----
Execute	AC ← MBR	101	1104	104	0023	0023

Inst: LOAD 104

What does the computer do in the FDE cycle of this simple program?

- Remember the fetch-decode-execute (FDE) cycle
- For the **first instruction** (LOAD 104) at address 100

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	----	----	----	----
Fetch	MAR ← PC	100	----	100	----	----
	IR ← M[MAR]	100	1104	100	----	----
	PC ← PC + 1	101	1104	100	----	----
Decode	MAR ← IR[11-0]	101	1104	104	----	----
	(Decode IR[15-12])	101	1104	104	----	----
Get operand	MBR ← M[MAR]	101	1104	104	0023	----
Execute	AC ← MBR	101	1104	104	0023	0023

Inst: LOAD 104

0023_h was in M[104]

What does the computer do in the FDE cycle of this simple program?

- Remember the fetch-decode-execute (FDE) cycle
- For the **first instruction** (LOAD 104) at address 100

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	----	----	----	----
Fetch	MAR ← PC	100	----	100	----	----
	IR ← M[MAR]	100	1104	100	----	----
	PC ← PC + 1	101	1104	100	----	----
Decode	MAR ← IR[11-0]	101	1104	104	----	----
	(Decode IR[15-12])	101	1104	104	----	----
Get operand	MBR ← M[MAR]	101	1104	104	0023	----
Execute	AC ← MBR	101	1104	104	0023	0023

Inst: LOAD 104

0023_h was in M[104]

What does the computer do in the FDE cycle of this simple program?

- Remember the fetch-decode-execute (FDE) cycle
- For the **first instruction** (LOAD 104) at address 100

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		100	----	----	----	----
Fetch	MAR ← PC	100	----	100	----	----
	IR ← M[MAR]	100	1104	100	----	----
	PC ← PC + 1	101	1104	100	----	----
Decode	MAR ← IR[11-0]	101	1104	104	----	----
	(Decode IR[15-12])	101	1104	104	----	----
Get operand	MBR ← M[MAR]	101	1104	104	0023	----
Execute	AC ← MBR	101	1104	104	0023	0023

Inst: LOAD 104

0023_h was in M[104]

What does the computer do in the FDE cycle of this simple program?

For the **second instruction**, i.e., ADD 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ← PC	101	1104	101	0023	0023
	IR ← M[MAR]	101	3105	101	0023	0023
	PC ← PC + 1	102	3105	101	0023	0023
Decode	MAR ← IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ← M[MAR]	102	3105	105	FFE9	0023
Execute	AC ← AC + MBR	102	3105	105	FFE9	000C

What does the computer do in the FDE cycle of this simple program?

For the **second instruction**, i.e., ADD 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ← PC	101	1104	101	0023	0023
	IR ← M[MAR]	101	3105	101	0023	0023
	PC ← PC + 1	102	3105	101	0023	0023
Decode	MAR ← IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ← M[MAR]	102	3105	105	FFE9	0023
Execute	AC ← AC + MBR	102	3105	105	FFE9	000C

What does the computer do in the FDE cycle of this simple program?

For the **second instruction**, i.e., ADD 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ← PC	101	1104	101	0023	0023
	IR ← M[MAR]	101	3105	101	0023	0023
	PC ← PC + 1	102	3105	101	0023	0023
Decode	MAR ← IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ← M[MAR]	102	3105	105	FFE9	0023
Execute	AC ← AC + MBR	102	3105	105	FFE9	000C

What does the computer do in the FDE cycle of this simple program?

For the **second instruction**, i.e., ADD 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ← PC	101	1104	101	0023	0023
	IR ← M[MAR]	101	3105	101	0023	0023
	PC ← PC + 1	102	3105	101	0023	0023
Decode	MAR ← IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ← M[MAR]	102	3105	105	FFE9	0023
Execute	AC ← AC + MBR	102	3105	105	FFE9	000C

What does the computer do in the FDE cycle of this simple program?

For the **second instruction**, i.e., ADD 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ← PC	101	1104	101	0023	0023
	IR ← M[MAR]	101	3105	101	0023	0023
	PC ← PC + 1	102	3105	101	0023	0023
Decode	MAR ← IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ← M[MAR]	102	3105	105	FFE9	0023
Execute	AC ← AC + MBR	102	3105	105	FFE9	000C

What does the computer do in the FDE cycle of this simple program?

For the **second instruction**, i.e., ADD 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ← PC	101	1104	101	0023	0023
	IR ← M[MAR]	101	3105	101	0023	0023
	PC ← PC + 1	102	3105	101	0023	0023
Decode	MAR ← IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ← M[MAR]	102	3105	105	FFE9	0023
Execute	AC ← AC + MBR	102	3105	105	FFE9	000C

What does the computer do in the FDE cycle of this simple program?

For the **second instruction**, i.e., ADD 105

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		101	1104	104	0023	0023
Fetch	MAR ← PC	101	1104	101	0023	0023
	IR ← M[MAR]	101	3105	101	0023	0023
	PC ← PC + 1	102	3105	101	0023	0023
Decode	MAR ← IR[11-0]	102	3105	105	0023	0023
	(Decode IR[15-12])	102	3105	105	0023	0023
Get operand	MBR ← M[MAR]	102	3105	105	FFE9	0023
Execute	AC ← AC + MBR	102	3105	105	FFE9	000C

What does the computer do in the FDE cycle of this simple program?

For the **third instruction**, i.e., STORE 106

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ← PC	102	3105	102	FFE9	000C
	IR ← M[MAR]	102	2106	102	FFE9	000C
	PC ← PC + 1	103	2106	102	FFE9	000C
Decode	MAR ← IR[11-0]	103	2106	106	FFE9	000C
	(Decode IR[15-12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ← AC	103	2106	106	000C	000C
	M[MAR] ← MBR	103	2106	106	000C	000C

What does the computer do in the FDE cycle of this simple program?

For the **third instruction**, i.e., STORE 106

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ← PC	102	3105	102	FFE9	000C
	IR ← M[MAR]	102	2106	102	FFE9	000C
	PC ← PC + 1	103	2106	102	FFE9	000C
Decode	MAR ← IR[11-0]	103	2106	106	FFE9	000C
	(Decode IR[15-12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ← AC	103	2106	106	000C	000C
	M[MAR] ← MBR	103	2106	106	000C	000C

What does the computer do in the FDE cycle of this simple program?

For the **third instruction**, i.e., STORE 106

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ← PC	102	3105	102	FFE9	000C
	IR ← M[MAR]	102	2106	102	FFE9	000C
	PC ← PC + 1	103	2106	102	FFE9	000C
Decode	MAR ← IR[11-0]	103	2106	106	FFE9	000C
	(Decode IR[15-12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ← AC	103	2106	106	000C	000C
	M[MAR] ← MBR	103	2106	106	000C	000C

What does the computer do in the FDE cycle of this simple program?

For the **third instruction**, i.e., STORE 106

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ← PC	102	3105	102	FFE9	000C
	IR ← M[MAR]	102	2106	102	FFE9	000C
	PC ← PC + 1	103	2106	102	FFE9	000C
Decode	MAR ← IR[11-0]	103	2106	106	FFE9	000C
	(Decode IR[15-12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ← AC	103	2106	106	000C	000C
	M[MAR] ← MBR	103	2106	106	000C	000C

What does the computer do in the FDE cycle of this simple program?

For the **third instruction**, i.e., STORE 106

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ← PC	102	3105	102	FFE9	000C
	IR ← M[MAR]	102	2106	102	FFE9	000C
	PC ← PC + 1	103	2106	102	FFE9	000C
Decode	MAR ← IR[11-0]	103	2106	106	FFE9	000C
	(Decode IR[15-12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ← AC	103	2106	106	000C	000C
	M[MAR] ← MBR	103	2106	106	000C	000C

What does the computer do in the FDE cycle of this simple program?

For the **third instruction**, i.e., STORE 106

Step	RTN	PC	IR	MAR	MBR	AC
(initial values)		102	3105	105	FFE9	000C
Fetch	MAR ← PC	102	3105	102	FFE9	000C
	IR ← M[MAR]	102	2106	102	FFE9	000C
	PC ← PC + 1	103	2106	102	FFE9	000C
Decode	MAR ← IR[11-0]	103	2106	106	FFE9	000C
	(Decode IR[15-12])	103	2106	106	FFE9	000C
Get operand	(not necessary)	103	2106	106	FFE9	000C
Execute	MBR ← AC	103	2106	106	000C	000C
	M[MAR] ← MBR	103	2106	106	000C	000C

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

A Simple Computer: Extending MARIE ISA

—Dr H. Asad

Extending Our Instruction Set

- So far, all of the MARIE instructions that we have discussed use a **direct addressing mode**.
- This means that the address of the operand is explicitly stated in the instruction.
- It is often useful to employ **indirect addressing**, where **the address of the address of the operand** is given in the instruction.
  - If you have ever used **pointers** in a program, you are already familiar with indirect addressing.

Extending the Instruction set

- Adding extra addressing modes

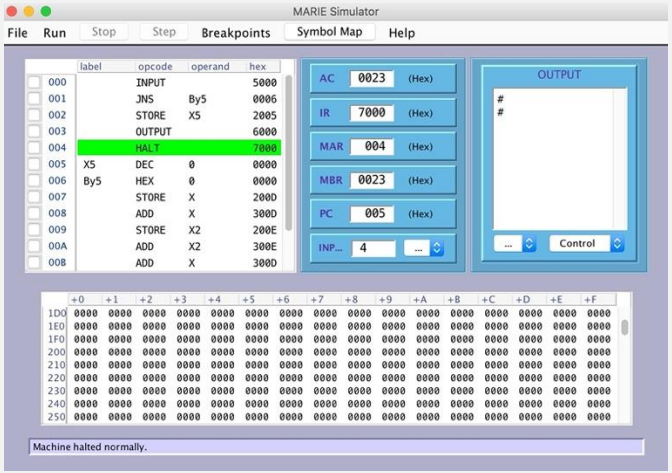
LOADI X	STOREI X	ADDI X
MAR ¬ X	MAR ¬ X	MAR ¬ X
MBR ¬ M[MAR]	MBR ¬ M[MAR]	MBR ¬ M[MAR]
MAR ¬ MBR	MAR ¬ MBR	MAR ¬ MBR
MBR ¬ M[MAR]	MBR ¬ AC	MBR ¬ M[MAR]
AC ¬ MBR	M[MAR] ¬ MBR	AC ¬ AC + MBR

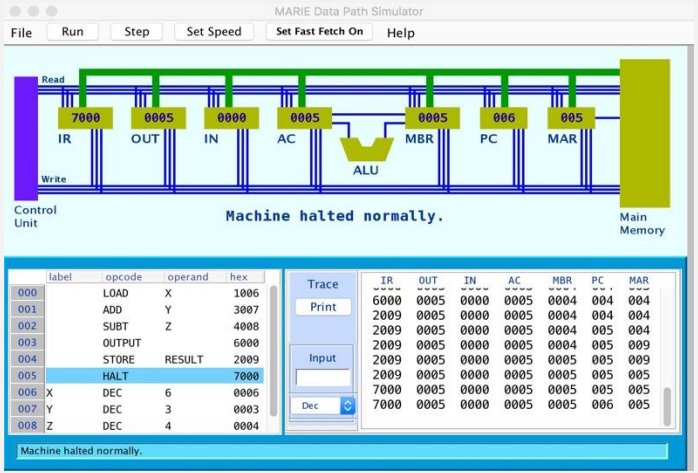
- The second operand gives the address of the operand to be loaded, stored or added

Extending the MARIE Instruction set

Instruction Number (hex)	Instruction	Meaning
0	JnS *X*	Store the PC at address *X* and jump to *X* + 1.
A	Clear	Put all zeros in AC.
B	AddI *X*	Add indirect: Go to address *X*. Use the value at *X* as the actual address of the data operand to add to AC.
C	JumpI *X*	Jump indirect: Go to address *X*. Use the value at *X* as the actual address of the location to jump to.
D	LoadI *X*	Load indirect: Go to address *X*. Use the value at *X* as the actual address of the operand to load into the AC.
E	StoreI *X*	Store indirect: Go to address *X*. Use the value at *X* as the destination address for storing the value in the accumulator.

Using MARIE





- You can use the MARIE simulator online:
  - https://marie.js.org
- Try some simple examples from this website and the book

Use of SKIPCOND

- Recall that **SKIPCOND** skips the next instruction according to the value of the AC
- The RTL for this instruction is the most complex in our instruction set:

```
If IR[11 - 10] = 00 then                                  (if bit 11 = 0 and bit 10 = 0)
            If AC < 0 then PC ¬   PC + 1

else If IR[11 - 10] = 01       then                       (if bit 11 = 0 and bit 10 = 1)

             If AC = 0 then PC ¬   PC + 1

else If IR[11 - 10] = 10       then                       (if bit 11 = 1 and bit 10 = 0)

             If AC > 0 then PC ¬   PC + 1
```

- Note: The notation **IR[11-10]** means the values of the 11th and 10th bit in register IR

A MARIE Program, Example

- Labels (e.g. Loop, Next) name to identify particular memory addresses
- If we allow the obvious meaning for labels, what does this code do?

Hex addr	label	instruction	
100	**If,**	**Load**	X
101		**Subt**	Y
102		Skipcond	**400**
103		Jump	Else
104	Then,	Load	X
105		Add	X
106		Store	X
107		Jump	Endif
108	Else,	Load	Y
109		Subt	X
10A		Store	Y
10B	Endif,	Halt	
10C	X,	Dec	12
10D	Y,	Dec	20

A MARIE Program, Example

- Labels (e.g. Loop, Next) name or identify particular memory addresses
- If we allow the obvious meaning for labels, what does this code do?

Hex addr	label	instruction	
100	**If,**	**Load**	X
101		**Subt**	Y
102		Skipcond	**400**
103		Jump	Else
104	Then,	Load	X
105		Add	X
106		Store	X
107		Jump	Endif
108	Else,	Load	Y
109		Subt	X
10A		Store	Y
10B	Endif,	Halt	
10C	X,	Dec	12
10D	Y,	Dec	20

```
if X = Y then
    X = X + X
else
    Y = Y - X
```

Opcode	Instruction	RTN
0000	JnS *X*	MBR ← PC MAR ← *X* M[MAR] ← MBR MBR ← *X* AC ← 1 AC ← AC + MBR PC ← AC
0001	Load *X*	MAR ← *X* MBR ← M[MAR], AC ← MBR
0010	Store *X*	MAR ← *X*, MBR ← AC M[MAR] ← MBR
0011	Add *X*	MAR ← *X* MBR ← M[MAR] AC ← AC + MBR
0100	Subt *X*	MAR ← *X* MBR ← M[MAR] AC ← AC - MBR
0101	Input	AC ← InREG
0110	Output	OutREG ← AC
0111	Halt	
1000	Skipcond	If IR[11–10] = 00 then If AC < 0 then PC ← PC + 1 Else If IR[11–10] = 01 then If AC = 0 then PC ← PC + 1 Else If IR[11–10] = 10 then If AC > 0 then PC ← PC + 1
1001	Jump *X*	PC ← IR[11–0]
1010	Clear	AC 0
1011	AddI *X*	MAR ← X MBR ← M[MAR] MAR ← MBR MBR ← M[MAR] AC ← AC + MBR
1100	JumpI *X*	MAR ← *X* MBR ← M[MAR] PC ← MBR

TABLE 4.7 **MARIE's Full Instruction Set**

The Full MARIE Instruction Set

- Here's the complete set of instructions for MARIE with their functionality
- Note: LoadI and StoreI is not given here but is part of the set.

A simple INPUT/OUTPUT program

- Write a program to sum 2 numbers given as INPUT and put the result in OUTPUT.

```
INPUT
STORE X  /some address outside the code space
INPUT
ADD X
OUTPUT
HALT
X, DEC 0
```

Exercises

- Write a MARIE program to add the two numbers from a memory location (pick sensible addresses) and store the result in memory
- Write a MARIE program which calculates the following expression

  INPUT – A + B

  and stores the result in C

CITY
ST GEORGE'S
UNIVERSITY OF LONDON

School of Science
& Technology

www.city.ac.uk

Systems Architecture

IN1006

A Simple Computer: CISC vs RISC

—Dr H. Asad

Two Very Different Design Philosophies

- Computers become more and more complex and their instruction sets reflect this. Two philosophies:
  - **Complex Instruction Set Computers (CISC)**
    - Each single instruction executes a number of low-level instructions
    - Multiple addressing modes
    - Pre-1990

- Computers are faster when the instruction set is simple since each instruction can be executed faster
  - **Reduced Instruction Set Computer (RISC)**
    - Each instruction is a single low-level instruction
    - Few addressing modes
    - Post-1990

RISC vs CISC

A CISC Architecture: Intel CPUs
A RISC Architecture: MIPS

- MIPS: being used in network routers, Smart TVs, video game consoles etc.

Question

- How a code written in C/Python/Java gets executed on the hardware? For example the following C statement:

```
If  x > y
    x=x-1;
 else
    y=y-x;
```

Summary

- A model of a simple computer
- MARIE
- Programming at a very low level

Hopefully you have a better understanding of what kinds of “code” computers actually execute

School of Science & Technology
City, University of London
Northampton Square
London
EC1V 0HB
United Kingdom

T: +44 (0)20 7040 5060
E: SST-ug@city.ac.uk
www.city.ac.uk/department

Credits:

Text and images for this lecture come from the recommended textbook (Chapter 4 MARIE: An Introduction to a Simple Computer The Essentials of Computer Organization and Architecture)