Tutorial 1
Intro and Background

CO 2103 Assembly Language

The tutorial is designed to make you think from hardware (a bit) and then moving into software (at low level). As a low-level programmer, you need sound knowledge of the hardware that you are using. The tutorial intends to let you appreciate the linkage between hardware and software, while applying what you have learned (if any) from the lectures. It intends to let you appreciate the significance of Machine Codes and hence Assembly Language programming.

Logic Circuit - 1

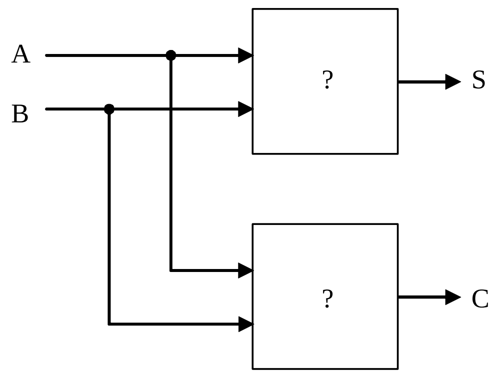
- Considering binary addition:
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 10
- Ignoring carry (only 1 bit result), consider the Augend (A) and Addend (B) as the inputs and the Sum (S) as output, we have

Inputs		*Output*
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

- **Task 1:** Design a logic circuit that will provide the function of 1-bit addition shown above. Hint: Refer to slides on Logic Gates.

Logic Circuit - 2

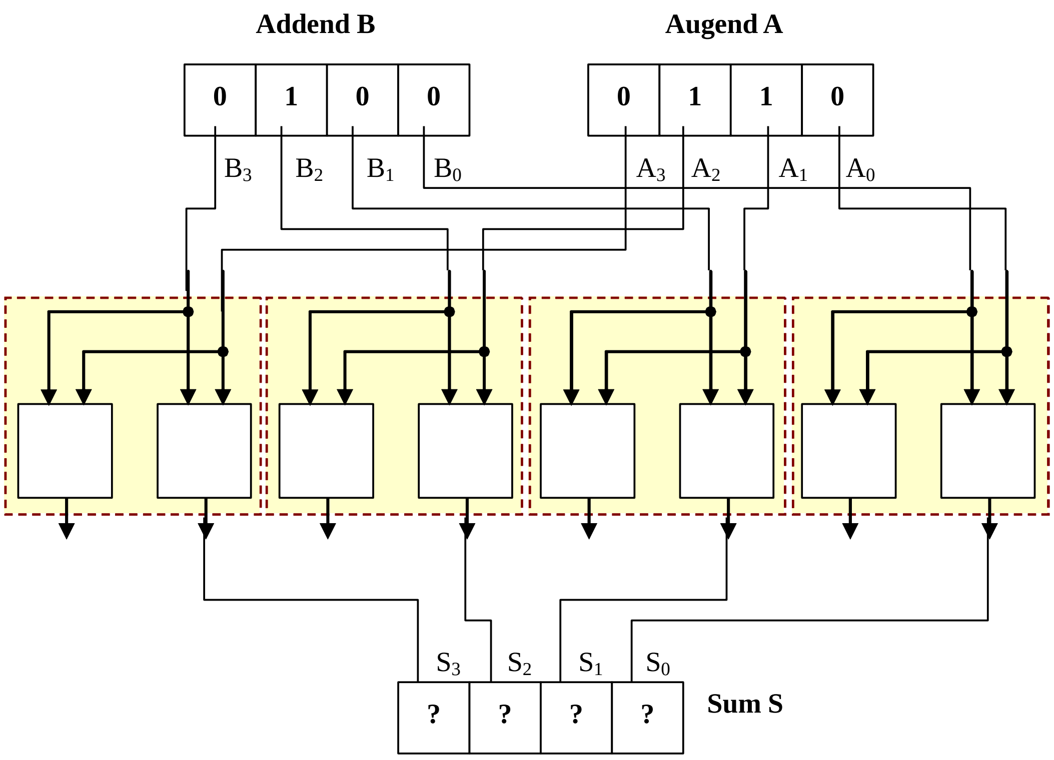
- The 1-bit Adder designed earlier and its truth table is not complete as it ignored the Carry
- **Task 2:** Draw the truth table for a 1-bit Adder with 2 inputs (Augend A and Addend B) and 2 outputs (Sum S and Carry C).
- **Task 3:** For the truth table in Task 2, design the logic circuit to provide the functions. Hint: Treat each output as an independent circuit.



- A 1-bit Adder adds two 1-bit numbers. For two n-bit numbers, we use n 1-bit Adders.

Logic Circuit - 3

- **Task 4:** Diagram on the right shows the use of four 1-bit Adder (from Task 3) to perform 4-bit hardware addition. Determine the Sum. Is the Sum correct? Explain what is incomplete and conclude that the 1-bit Adder in Task 3 is a Half-Adder.



Logic Circuit - 4

- **Task 5:** A Full-Adder will have 3 inputs – Augend A, Addend B and Carry In C_i, and have 2 outputs – Sum S and Carry Out C_o. Draw the Truth Table for a Full-Adder and design a Full-Adder using two Half-Adders. Hint: Use an OR gate to combine the Carries from both Half-Adders to give you the final Carry Out.
- **Task 6:** Replace the Half-Adders in the diagram in Task 4 with Full-Adders and make necessary connections. Draw the Full-Adder as a Box with 3 inputs and 2 outputs. Determine the Sum. Is the Sum correct?

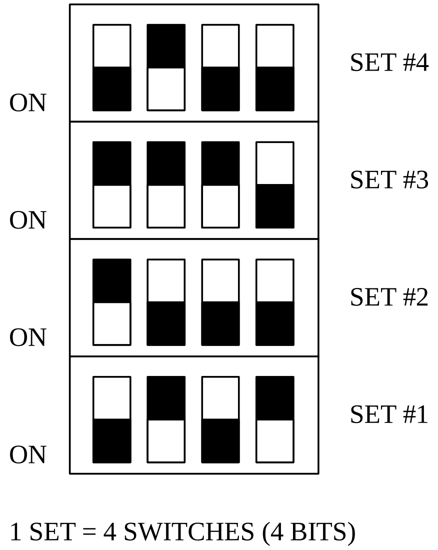
Interconnecting Logic Gates gives Logic Circuit with simple desired function, while complex functions can be achieved by interconnecting Logic Circuits. A microprocessor is built from complex interconnections of Logic Circuits; giving a versatile (various functions) but complex Logic Circuit. More on this in CO2206 …

Hardware to Software - 1

- Where do those zeros and ones come from? In hardware, data are stored as ON or OFF. This can be achieved through storage of electric charge. We can analogize this concept as having hardware switches to store our data. We can imagine that each set of switches is a data storage or memory location.

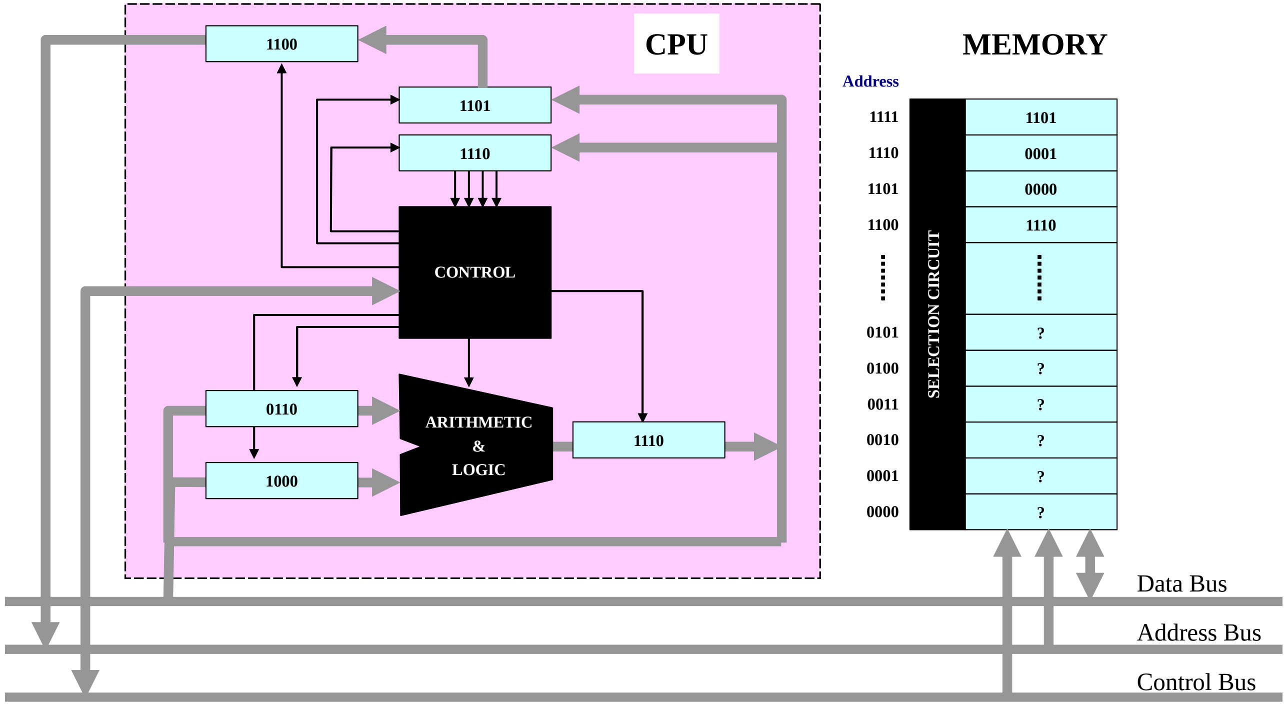
- **Task 7:** Diagram on the right shows four set of switches used to store our data. Let ON=1 and OFF=0, determine the data (in binary form) stored in each set. Note "down" position is ON and black box indicates position of the switch.

- **Task 8:** How many different objects (e.g. character, word, statement, instruction, etc) can be represented by a 4-bit code?



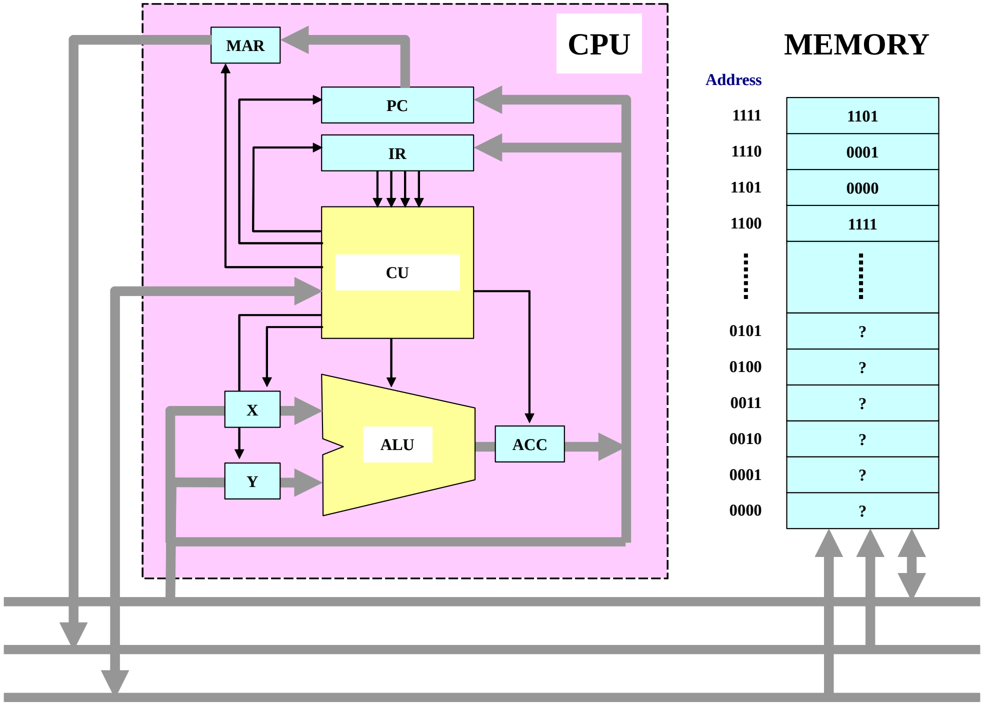
Hardware to Software - 2

- Diagram below shows an over simplified hypothetical CPU. The light blue boxes (with zeros and ones) are set of electronic switches making up the data storage. The black boxes are complex electronic logic circuits designed for their specific function(s).



Hardware to Software - 3

- The hypothetical CPU in previous slide handles 4-bit Data and 6 registers (data storage: X, Y, ACC, IR, PC, MAR). It can access up to 16 memory locations (only). Data are stored into the memory locations through hardware (e.g. hardwired, switches) or from registers (only).



CU = Control Unit
ALU = Arithmetic and Logic Unit

Hardware to Software - 4

- The Instruction Set for the system in previous slide is shown on the right. As an example, the low level (Assembly Language) implementation for the high level statement of

  F = A + B

  (where F is stored in location 1100, A and B are content of location 1111 and 1110 respectively) will be as follow:

  GETX3 ; X=A
  GETY2 ; Y=B
  ADD ; ACC=A+B
  PUTA0 ; F=ACC=A+B

Machine Code	Instruction	Function
0000	GETX0	(X) ← (1100)
0001	GETX1	(X)← (1101)
0010	GETX2	(X) ← (1110)
0011	GETX3	(X) ← (1111)
0100	GETY0	(Y) ← (1100)
0101	GETY1	(Y) ← (1101)
0110	GETY2	(Y) ← (1110)
0111	GETY3	(Y) ← (1111)
1000	PUTA0	(1100) ← (ACC)
1001	PUTA1	(1101) ← (ACC)
1010	PUTA2	(1110) ← (ACC)
1011	PUTA3	(1111) ← (ACC)
1100	CLRX	(X) = 0
1101	CLRA	(ACC) = 0
1110	ADD	ACC ← X + Y
1111	SUB	ACC ← X – Y

Legend:

(R) Content of Register R (R can be X, Y or ACC)

(nnnn) Content of location nnnn

Hardware to Software - 5

- Task 9 to 13 refer to the system in the previous two slides.
- **Task 9:** Implement the following high level statement in low level.

  F = A – (B + C)

  (where F is stored in location 1100, A, B and C are content of location 1111, 1110 and 1101 respectively)
- Note at high level we are not concerned on where A, B, C and F are handled in the computer. However, at low level we have to be specific where are these variables stored in the memory (hardware).
- **Task 10:** Convert the AL program in Task 9 into Machine Language (Machine Codes) program.
- **Task 11:** If the CPU will start executing from location 0000 every time it is turned ON, determine where will the Machine Codes be stored in the memory.

Hardware to Software - 6

- **Task 12:** Using the data in Slide 8, determine the value stored at location 1100 after execution of the following instructions:

  GETX0
  GETY3
  SUB
  PUTA0

- The system investigated in Task 9 to 12 is not really useful as it does not allow us to store external data into the memory. For example, we can't perform the following operations (through software):
  - store 5 into location 1101
  - perform F = 3 + 10
- **Task 13**: State three other operations the system cannot perform.
- To enable the above operations, the Instruction Set need to be modified - hence hardware logic to be modified. We will leave this challenging issue to an assessed work in near future.

Hardware to Software - 7

- In Tasks 7 to 13, you worked from hardware (including hardware data – switches) into the computer domain (microprocessor-based system).
- The Instruction Set (hence the complication of its design – data size, architecture) determines the functions/operations it can achieve (its limitations)
- CPU only handles zeros and ones – and so everything else must be represented by zeros and ones … moving us into the number systems and data representation.

Number Systems - 1

- **Task 14:**
  1. Convert the following binary numbers to decimal::
     (a) 0110, (b) 1011, (c) 11110000, (d) 10101010
  2. Convert the following binary numbers to hexadecimal:
     (a) 1110, (b) 11011, (c) 110110101, (d) 1010111101110010
  3. Convert the following decimal numbers to binary and hexadecimal:
     (a) 12, (b) 15, (c) 27, (d) 96
  4. Perform the following unsigned binary additions:
     (a) 1 + 1, (b) 1010 + 1111, (c) 110111 + 11001
  5. If a program variable is to be used to store a unique number identifying any day in the year, how many bits will be required to store it? How many bits to store the year?

Number Systems - 2

- **Task 14:**

6. Perform the following unsigned binary subtractions:
   - 100101 – 1111, 11100 – 1010, 101011 – 1110, 1100 – 11
7. Perform the following hexadecimal subtractions:
   - 123 – DD, 3FF – 20, AB00 – 123, A200 – 3FD

Signed Integers Representation

- **Task 15:**
  1. For an 10-bit group, work out the representation for –371 in
     (a) Sign & Magnitude, (b) 1's Complement, (c) 2's Complement,
     (d) Excess-512, (e) Excess-400
  2. For a 10-bit group, what range of integers can be represented using
     (a) Sign & Magnitude, (b) 1's Complement, (c) 2's Complement,
     (d) Excess-512
  3. Express 9876510 in BCD
  4. Form the negative equivalent of the following 8-bit 2's Complement numbers
     (a) 00011001, (b) 00011110, (c) 01101000, (d) 01110100
     by comparing the resulting bit patterns to the originals, can you spot a "short cut" method for the conversion? Hint: Change Sign Rule III
  5. Perform the following 12-bit 2's complement subtraction
     1010 1010 1011 – 1011 0000 1101

ASCII

- **Task 16:** Referring to ASCII code table, determine the message stored in the memory as shown below with the first character starting at lowest memory location 1000 0000 (80_h).

Address:		Address:		Address:		Address:		Address:	
1000 0000	0100 0010	1000 1100	0110 0001	1001 1000	0110 0100	1010 0100	0111 0010	1011 0000	0110 1111
1000 0001	0111 0010	1000 1101	0111 0110	1001 1001	0010 0000	1010 0101	0111 0011	1011 0001	0111 0010
1000 0010	0110 0001	1000 1110	0110 0101	1001 1010	0111 0100	1010 0110	0010 0000	1011 0010	0010 0000
1000 0011	0111 0110	1000 1111	0010 0000	1001 1011	0110 1000	1010 0111	0111 0100	1011 0011	0110 1111
1000 0100	0110 1111	1001 0000	0110 0011	1001 1100	0110 0101	1010 1000	0110 0111	1011 0100	0110 0110
1000 0101	0010 0001	1001 0001	0110 1111	1001 1101	0010 0000	1010 1001	0010 0000	1011 0101	0010 0000
1000 0110	0010 0000	1001 0010	0110 1110	1001 1110	0110 0010	1010 1010	0111 0100	1011 0110	0100 0001
1000 0111	0111 1001	1001 0011	0111 0001	1001 1111	0110 0001	1010 1011	0110 1000	1011 0111	0100 1100
1000 1000	0110 1111	1001 0100	0111 0101	1010 0000	0111 0010	1010 1100	0110 0101	1011 1000	0010 0000
1000 1001	0111 0101	1001 0101	0110 0101	1010 0001	0111 0010	1010 1101	0010 0000	1011 1001	0010 1110
1000 1010	0010 0000	1001 0110	0111 0010	1010 0010	0110 1001	1010 1110	0110 0100	1011 1010	0010 1110
1000 1011	0110 1000	1001 0111	0110 0101	1010 0011	0110 0101	1010 1111	0110 1111	1011 1011	0010 1110

- Note a short hand to write the above memory contents is in HEX:
80: 42 72 61 76 6F 21 20 79 6F 75 20 68 61 76 65 20 63 6F 6E 71
94: 75 65 72 65 64 20 74 68 65 20 62 61 72 72 69 65 72 73 20 74
A8: 6F 20 74 68 65 20 64 6F 6F 72 20 6F 66 20 41 4C 20 2E 2E 2E