Computational Microelectronics L1

Sung-Min Hong

smhong@gist.ac.kr

Semiconductor Device Simulation Laboratory, GIST

Welcome

Welcome!

- Computational Microelectronics
  - Code: EC7114 (/ SE7114)
  - Lecture 3, no experiment, credit 3

- Instructor: Sung-Min Hong
  - School of EECS

Resources

- Presentation materials

  https://github.com/hi2ska2/cm2024f

- Homework submission
  - GIST LMS system

- YouTube channel

  https://www.youtube.com/c/SungMinHong

Evaluation

- Attendance (10%)
- Homework (40%)
- Final presentation (50%)
  - Prepare and submit your own presentation.
  - It will be uploaded in my YouTube channel.

Planned business trips

- Three weeks
  - Bruges (September 8~14. ESSERC)
  - San Jose (September 23~29. SISPAD)
  - San Francisco (December. IEDM)
- Pre-recorded lectures will be uploaded.

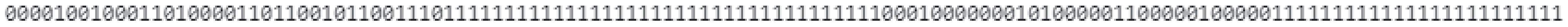
SAP(Simple-As-Possible)-1 instruction

Input file

- Assume that your input file has the following format:
  - 16 X 8 binary numbers. For example, it may read

```
0000 1001 0001 1010 0001 1011 0010 1100 1110 1111
1111 1111 1111 1111 1111 1111 1111 1111 0001 0000
0001 0100 0001 1000 0010 0000 1111 1111 1111 1111
1111 1111
```

  - Of course, your actual input file contains a long bit stream. (You can find it in our GitHub repository.)



```
00001001000110100001101100101100111011111111111111111111111111111111111100010000000101000001100000100000111111111111111111111111
```

Meaning of each row

- There are 16 rows.
  - Process each row sequentially.

$$\text{Most significant bit} \rightarrow \underbrace{0000}_{\text{Instruction field}}\ \underbrace{1001}_{\text{Address field}} \leftarrow \text{Least significant bit}$$

0000 (Load the accumulator. LDA)

0001 (Add. ADD)

0010 (Subtract. SUB)

1110 (Transfer the accumulator contents to the output port. OUT)

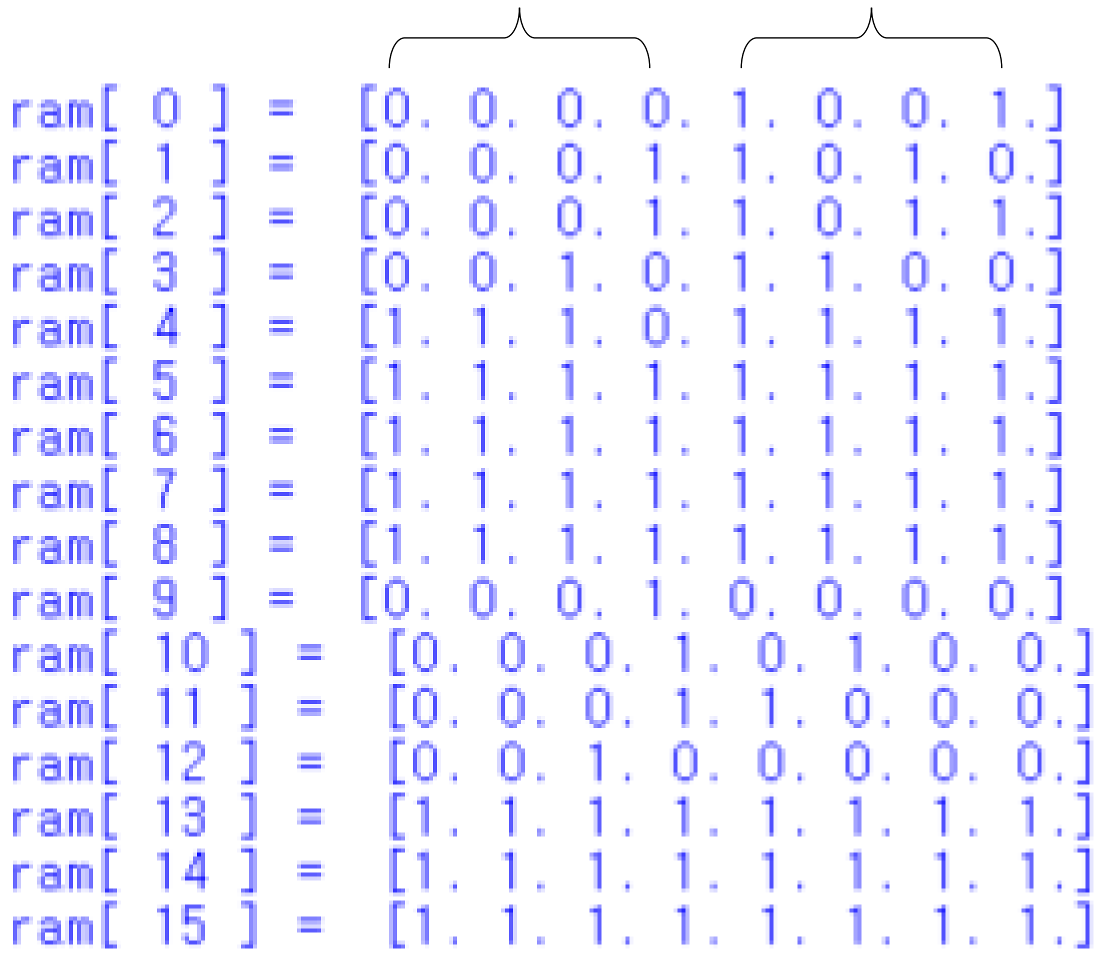
1111 (Halt. HLT)

Variables

- The following variables will be used.
  - ram: It is a 16 X 8 array. The input file is stored.
  - pc: It changes from 0 to 15.
  - regA
  - regB
  - sum
  - diff

RAM

- Write a program to read the input file.



```
ram[ 0 ] =  [0. 0. 0. 0. 1. 0. 0. 1.]
ram[ 1 ] =  [0. 0. 0. 1. 1. 0. 1. 0.]
ram[ 2 ] =  [0. 0. 0. 1. 1. 0. 1. 1.]
ram[ 3 ] =  [0. 0. 1. 0. 1. 1. 0. 0.]
ram[ 4 ] =  [1. 1. 1. 0. 1. 1. 1. 1.]
ram[ 5 ] =  [1. 1. 1. 1. 1. 1. 1. 1.]
ram[ 6 ] =  [1. 1. 1. 1. 1. 1. 1. 1.]
ram[ 7 ] =  [1. 1. 1. 1. 1. 1. 1. 1.]
ram[ 8 ] =  [1. 1. 1. 1. 1. 1. 1. 1.]
ram[ 9 ] =  [0. 0. 0. 1. 0. 0. 0. 0.]
ram[ 10 ] =  [0. 0. 0. 1. 0. 1. 0. 0.]
ram[ 11 ] =  [0. 0. 0. 1. 1. 0. 0. 0.]
ram[ 12 ] =  [0. 0. 1. 0. 0. 0. 0. 0.]
ram[ 13 ] =  [1. 1. 1. 1. 1. 1. 1. 1.]
ram[ 14 ] =  [1. 1. 1. 1. 1. 1. 1. 1.]
ram[ 15 ] =  [1. 1. 1. 1. 1. 1. 1. 1.]
```

0000 1001

- LDA
  - The address is 1001.
  - Then, regA is loaded with the contents of memory location at 1001 (`0001 0000`).
  - It means that regA now becomes a binary number of 0001 0000 (a decimal number of 16).

0001 1010, 0010 1100

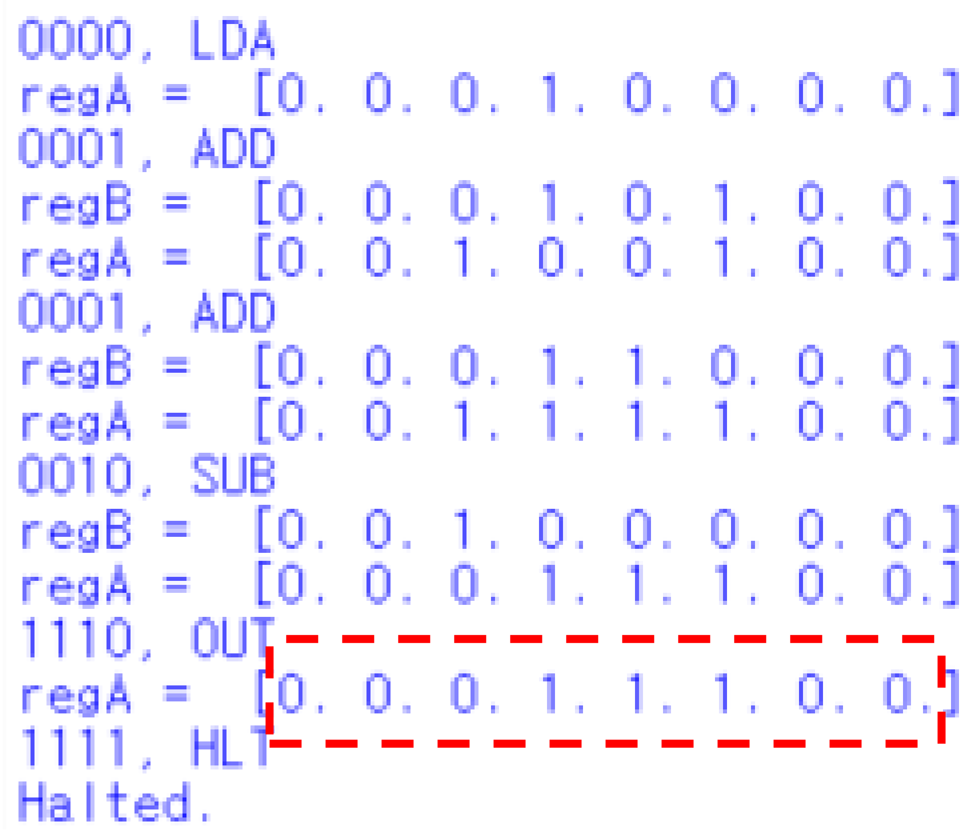
- ADD
  - The address is 1010.
  - First, regB is loaded with the contents of memory location at 1010 (`0001 0100`).
  - Then, regA becomes regA + regB. For example, regA now becomes a binary number of 0010 0100.
- SUB
  - The address is 1100.
  - First, regB is loaded with the contents of memory location at 1100 (`0010 0000`).
  - Then, regA becomes regA – regB.

1110 XXXX, 1111 XXXX

- OUT
  - The address, XXXX, does not matter. We do not care about it.
  - Print out the content of regA (in the binary number format).
  - In this example, the output will be `0001 1100`.
- HLT
  - The program is stopped.

Screen out

- Write your own program.



```
0000, LDA
regA =  [0. 0. 0. 1. 0. 0. 0. 0.]
0001, ADD
regB =  [0. 0. 0. 1. 0. 1. 0. 0.]
regA =  [0. 0. 1. 0. 0. 1. 0. 0.]
0001, ADD
regB =  [0. 0. 0. 1. 1. 0. 0. 0.]
regA =  [0. 0. 1. 1. 1. 1. 0. 0.]
0010, SUB
regB =  [0. 0. 1. 0. 0. 0. 0. 0.]
regA =  [0. 0. 0. 1. 1. 1. 0. 0.]
1110, OUT
regA =  [0. 0. 0. 1. 1. 1. 0. 0.]
1111, HLT
Halted.
```

Homework#1

- Due: AM08:00, September 5
- Problem#1
  - Run the following input file (hw1.inp). What happens?

```
0000 1111 0001 1110 1110 1111 0000 1101 1110 1111
1111 1111 1111 1111 1111 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 0000 0100 0000 0011
0000 0010
```

Thank you for your attention!