COMSM1302
Overview of Computer Architecture

Lecture 10

Introduction to assembly programming

In the previous lectures

- 4-bit CPU
- ModuleSim simulation software

In this lecture

Instruction set → Instruction cycle → Assembly programming problem

- At the end of this lecture:
  - Familiar with the concept of instruction set
  - Able to read, execute by hand and write assembly programs

From instructions to assembly code

Instruction set → Instruction cycle → Assembly programming problem

Load registers with a constant

1. LDAC Operand :
   – A <- Operand
   – LDAC opcode is 0000
2. LDBC Operand :
   – B <- Operand
   – LDBC opcode is 0001

Mnemonic	Machine code	After execution
LDAC 0x5	**0000**0101	A = 0x5
LDAC 0x6	**0000**0110	A = 0x6
LDBC 0x5	**0001**0101	B = 0x5
LDBC 0x6	**0001**0110	B = 0x6

Load registers with a value from memory

- LDAM Operand :
  - A <- [Operand]
  - LDAM opcode is 0010
- LDBM Operand :
  - B <- [Operand]
  - LDBM opcode is 0011

Mnemonic	Machine code	After execution
LDAM 0x5	**0010**0101	A = 0x1
LDAM 0x6	**0010**0110	A = 0x9
LDBM 0x5	**0011**0101	B = 0x1
LDBM 0x6	**0011**0110	B = 0x9

Address	value
0x5	0x01
0x6	0xf9

Store A register in memory

- STAM Operand :
  - $[\text{Operand}]_{0..3}$ <- A
  - $[\text{Operand}]^{4..7}$ <- 0000
  - STAM opcode is 0100

Before execution

Address	value
0x5	11110011
0x6	01101010

After execution

Address	value
0x5	00000001
0x6	00001001

Mnemonic	Machine code	Assume
STAM 0x5	**0100**0101	A = 0x1
STAM 0x6	**0100**0110	A = 0x9

ADD

- ADD (no operand):
  - A <- A + B
  - ADD opcode is 0101

Mnemonic	Machine code	Assume before execution	After execution
ADD	**0101**xxxx	A = 0x1 , B = 0x5	A = 0x6
ADD	**0101**xxxx	A = 0x9, B = 0x8	A = 0x1

SUB

- SUB (no operand):
  - A <- A - B
  - SUB opcode is 0110

Mnemonic	Machine code	Assume before execution	After execution
SUB	**0110**xxxx	A = 0x1 , B = 0x5	A = 0xC
SUB	**0110**xxxx	A = 0x9, B = 0x8	A = 0x1

Load A register from memory with offset

- LDAI Operand :
  - A <- [A + Operand]
  - LDAI opcode is 0111

Address	value
0x5	xxxx0011
0x6	xxxx1000

Mnemonic	Machine code	Assume before execution	After execution
LDAI 0x4	**0111**0100	A = 0x1	A = 0x3
LDAI 0xD	**0111**1101	A = 0x9	A = 0x8

Our instruction set

Machine code	Mnemonic	Description	Example	
0000	LDAC	Load register A with a constant value.	00000101	Load A with 5
0001	LDBC	Same as LDAC, but for B	00010110	Load B with 6
0010	LDAM	Load A with the value stored in the memory addressed by the operand.	00100101	Load A with the content of the 5th byte.
0011	LDBM	Same as LDAM, but for B	00110110	Load B with the content of the 6th byte.
0100	STAM	Store A reg in the memory location addressed by the operand and reset the four MSBs of this location.	01001010	Store A in the 10th byte of the memory and reset the 4 MSBs of this byte.
0101	ADD	A <- A + B	0101xxxx	
0110	SUB	A <- A - B	0110xxxx	
0111	LDAI	A <- [A + operand]	01110101	Load A with the address calculated from A + 5.
1111	???	???	???	???

First proper assembly code

1. What are the values of the registers A,B,O and memory location 0x06 after executing this code?

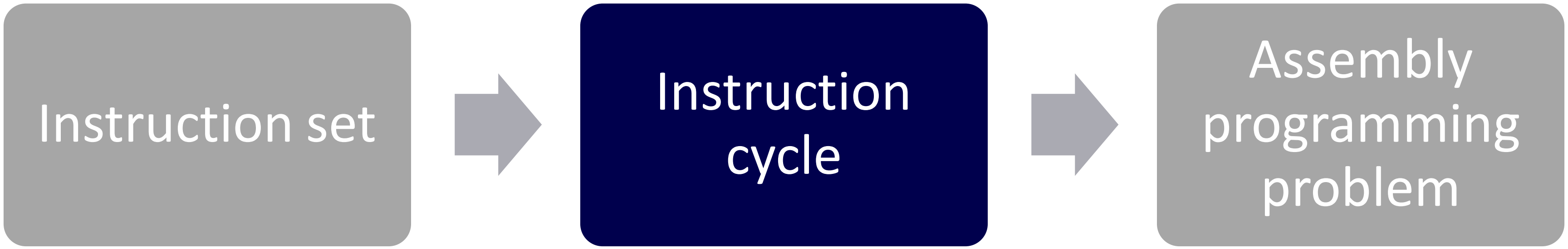
   0

2. What is the mnemonic of the machine code 0x54 ?

   ADD

Mnemonic	Address	Machine code
LDAC 0x3	0x0	0x03
LDBC 0x6	0x1	0x16
ADD	0x2	0x50
LDBC 0x7	0x3	0x17
ADD	0x4	0x50
STAM 0x6	0x5	0x46

From instructions to assembly code



Execution steps

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.

2. Increase the PC to remember what is the next instruction you need to fetch.

3. Decode the instruction.

4. Execute the instruction.

Assembly code execution -1/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000						
0x1	00010110							
0x2	01010000							
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -2/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011				
0x1	00010110							
0x2	01010000							
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -3/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001			
0x1	00010110							
0x2	01010000							
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -3/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode
0x0	00000011	0000	0000
0x1	00010110		
0x2	01010000		
0x3	00010111		
0x4	01010000		
0x5	01000110		
0x6	xxxxxxxx		

Machine code	Mnemonic	Description
0000	LDAC	Load register A with a constant value.
0001	LDBC	Same as LDAC, but for B
0010	LDAM	Load A with the value stored in the memory addressed by the operand.
0011	LDBM	Same as LDAM, but for B
0100	STAM	Store the value of A in the memory location addressed by the operand
0101	ADD	A <- A + B
0110	SUB	A <- A - B
0111	LDAI	A <- [A + operand]
1111	???	???

Assembly code execution -4/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001	LDAC 0x3		
0x1	00010110							
0x2	01010000							
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -5/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001	LDAC 0x3	0011	
0x1	00010110							
0x2	01010000							
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -6/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001	LDAC 0x3	0011	
0x1	00010110	0001						
0x2	01010000							
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -7/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001	LDAC 0x3	0011	
0x1	00010110	0001	0001	0110	0010	LDBC 0x3	0011	0110
0x2	01010000							
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -8/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode
0x0	00000011	0000	0000
0x1	00010110	0001	0001
0x2	01010000	0010	0101
0x3	00010111		
0x4	01010000		
0x5	01000110		
0x6	xxxxxxxx		

Instruction

Machine code	Mnemonic	Description
0000	LDAC	Load register A with a constant value.
0001	LDBC	Same as LDAC, but for B
0010	LDAM	Load A with the value stored in the memory addressed by the operand.
0011	LDBM	Same as LDAM, but for B
0100	STAM	Store the value of A in the memory location addressed by the operand
0101	ADD	A <- A + B
0110	SUB	A <- A - B
0111	LDAI	A <- [A + operand]
1111	???	???

Assembly code execution -9/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001	LDAC 0x3	0011	
0x1	00010110	0001	0001	0110	0010	LDBC 0x6	0011	0110
0x2	01010000	0010	0101	0000	0011	ADD	1001	0110
0x3	00010111							
0x4	01010000							
0x5	01000110							
0x6	xxxxxxxx							

Assembly code execution -10/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

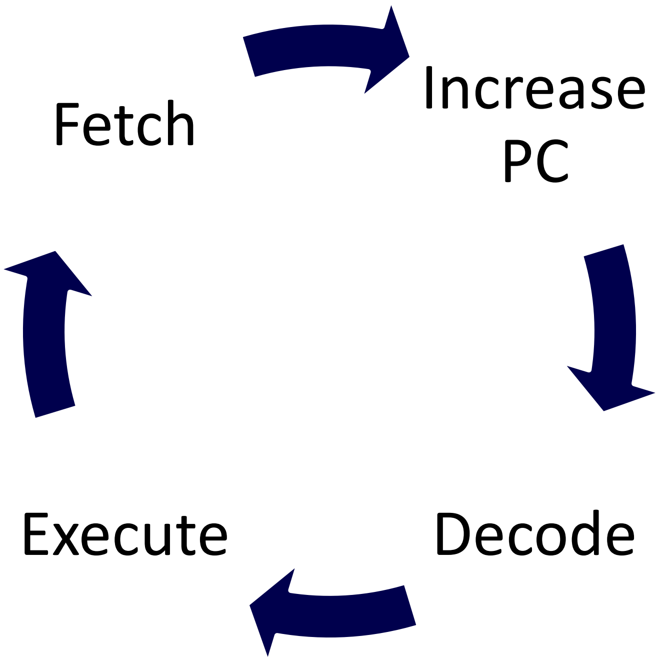
Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001	LDAC 0x3	0011	
0x1	00010110	0001	0001	0110	0010	LDBC 0x6	0011	0110
0x2	01010000	0010	0101	0000	0011	ADD	1001	0110
0x3	00010111	0011	0001	0111	0100	LDBC 0x7	1001	0111
0x4	01010000	0100	0101	0000	0101	ADD	0000	0111
0x5	01000110	0101	0100	0110	0110	STAM 0x6		
0x6	xxxxxxxx							

Assembly code execution -11/11

1. Fetch the instruction pointed by the PC from the memory to the Opcode and Operand.
2. Increase the PC to remember what is the next instruction you need to fetch.
3. Decode the instruction.
4. Execute the instruction.

Address	Data	PC current	Opcode	Operand	Next PC	Mnemonic	A	B
0x0	00000011	0000	0000	0011	0001	LDAC 0x3	0011	
0x1	00010110	0001	0001	0110	0010	LDBC 0x6	0011	0110
0x2	01010000	0010	0101	0000	0011	ADD	1001	0110
0x3	00010111	0011	0001	0111	0100	LDBC 0x7	1001	0111
0x4	01010000	0100	0101	0000	0101	ADD	0000	0111
0x5	01000110	0101	0100	0110	0110	STAM 0x6	0000	0111
0x6	**0000**0000							

Instruction cycle



What do we need to execute a code?

1. Memory dump.
2. Initial PC value.
3. Instruction Set.
4. Initial registers values.

From instructions to assembly code

Instruction set → Instruction cycle → **Assembly programming problem**

Assembly programming problem

- Write an assembly code using our ISA to calculate the difference between two two-digit numbers x and y.

1. After executing this program, the "A" register should contain the value $(x - y)$.
2. Your program should calculate correct answer for inputs that satisfy the following conditions:

   $0 \leq x - y \leq 15, 0 \leq x \leq 99, 0 \leq y \leq 99$, and the ones and tens of x are greater or equal to the ones and tens of y, respectively.

Plan

1. Some examples.
2. Store input data in the memory.
3. Discuss and analysis the problem.
4. Express the algorithm of our code as a flowchart.
5. Translate the operations in the flowchart to assembly instructions.
6. Sort out any memory issues.

Valid inputs example

1. Let the first two-digit number x be 95
   - $0 \leq x \leq 99$
2. Let the second two-digit number y be 81
   - $0 \leq y \leq 99$
3. $0 \leq x - y \leq 15$
4. The ones of $\geq$ the ones of y
5. The tens of $\geq$ the tens of y

Invalid input example

1. Let the first two-digit number x be 95
   - $0 \leq x \leq 99$
2. Let the second two-digit number y be 89
   - $0 \leq y \leq 99$
3. $0 \leq x - y \leq 15$
4. The ones of x $\geq$ the ones of y
5. The tens of x $\geq$ the tens of y

Thinking questions

- Remember our PC is 4 bits wide. What could be the maximum size of our code?

  16 bytes

- How and where x and y will be stored before we start our program?

  registers

- Remember the operands of our instructions are 4 bits wide only. How can we store numbers from 0 to 99 in the memory?

  use two registers to store the tens digits and ones digits

Binary-coded decimal

Decimal digit	BCD			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Difference between Hex and BCD

- What is the value 10 in hex?

0xA

- What is the value 10 in BCD?

$\underline{0001}\,|\,\underline{0000}$

Store inputs in memory – 1/3

- Let x = 95 and y = 81
- Using BCD: x = 1001 0101 , y = 1000 0001

- Our Operand is the least significant nibble of our instruction (bits 0 through 3)

Store inputs in memory – 2/3

- Let x = 95 and y = 81
- Let $x = x_0 + x_1 * 10$
- Let $y = y_0 + y_1 * 10$.

$95 = 5 + 10 \times 9$

- What are the values of x_0 x_1 y_0, and y_1 in decimal and BCD?

Store inputs in memory – 3/3

- Let x = 95 and y = 81
- $x_0 = 0101$, $x_1 = 1001$, $y_0 = 0001$, and $y_1 = 1000$
- Now we can store these values in the memory and they can be accessed by our CPU.

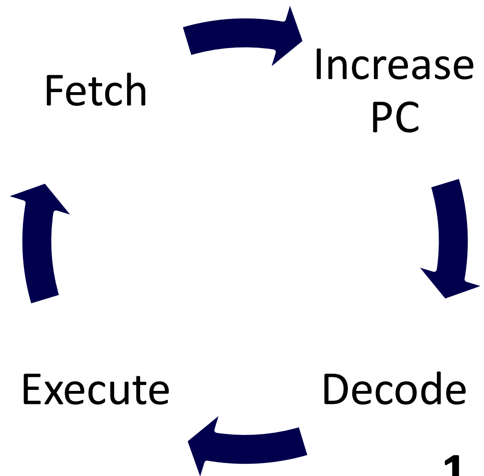
Address	value
0xC	0x05
0xD	0x09
0xE	0x01
0xF	0x08

Plan

1. Some examples.
2. Store input data in the memory.
3. Discuss and analysis the problem.
4. Express the algorithm of our code as a flowchart.
5. Translate the over code operations to assembly instructions.
6. Sort out any memory issues.

Summary

Mnemonic	Address	Machine code
LDAC 0x3	0x0	0x03
LDBC 0x6	0x1	0x16
ADD	0x2	0x50
LDBC 0x7	0x3	0x17
ADD	0x4	0x50
STAM 0x6	0x5	0x46



Address	value
0xC	0x05
0xD	0x09

- Write an assembly code using our ISA to calculate the difference between two two-digit numbers x and y.

1. **Memory dumb.**
2. **Initial PC value.**
3. **Instruction Set.**
4. **Initial registers values.**