Microprocessor（Lecture 1）

Introduction

- Portal for students' experiments (情報・知能工学系学生実験サイト)

  http://www.cs.tut.ac.jp/jikken/
- Documents are available at https://expcs.github.io/microprocessor/
- Reports should be submitted by e-mail to fukumura@cs.tut.ac.jp
  - レポートは日本語でも英語でも可
- If you have questions, you can e-mail me or visit F-413.

Schedule (see p. 26)

Week 1

Lecture 1: Introduction
Problem 3.1: Addition
Problem 3.3 (1): Single tone

Week 2

Lecture 2: Basic Programming
Problem 3.2: Multiplication

Week 3

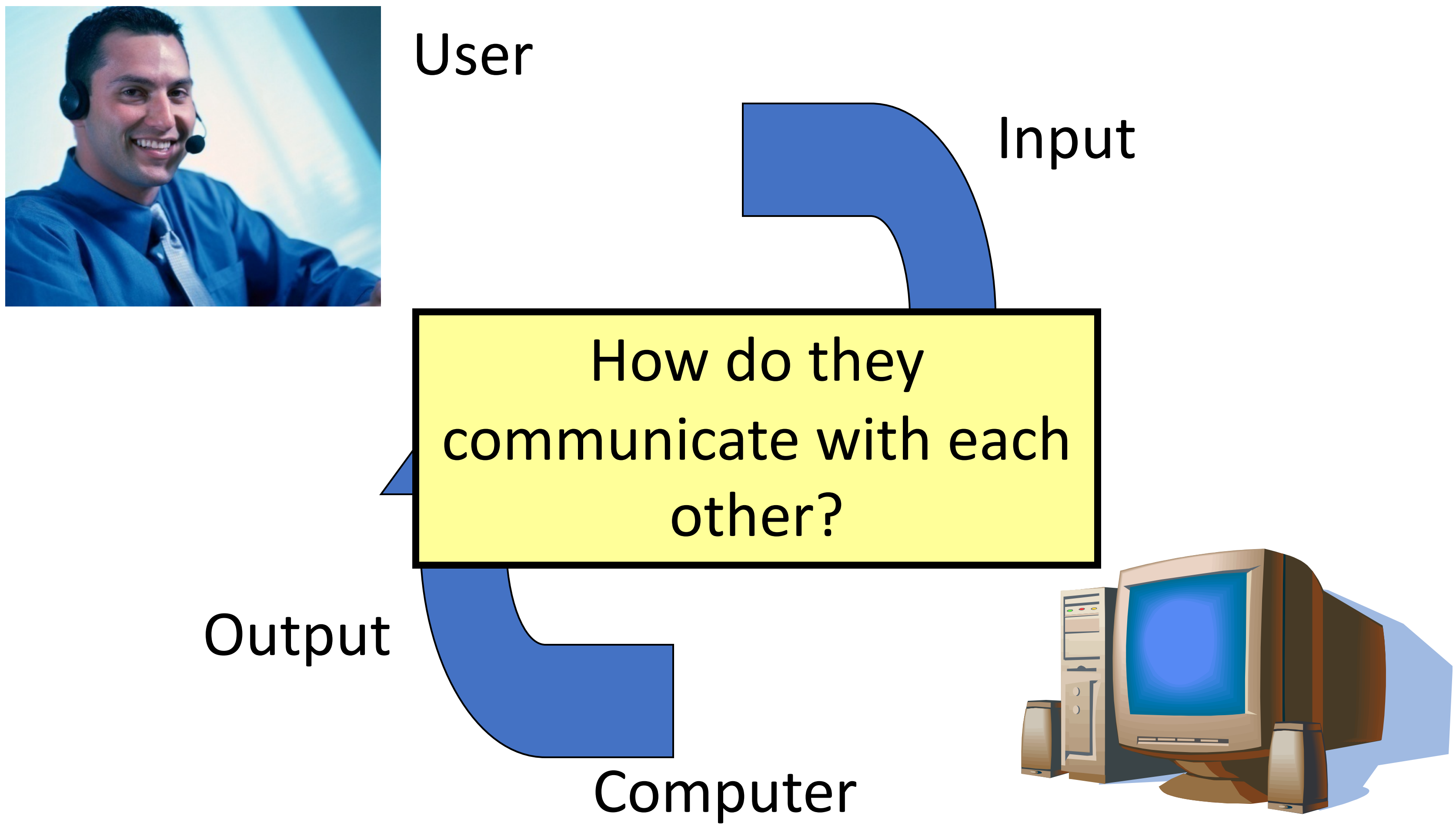
Lecture 4: Applied programming
Problem 3.3 (2): Melody

You should prepare programs for Problem 3.2 and 3.4

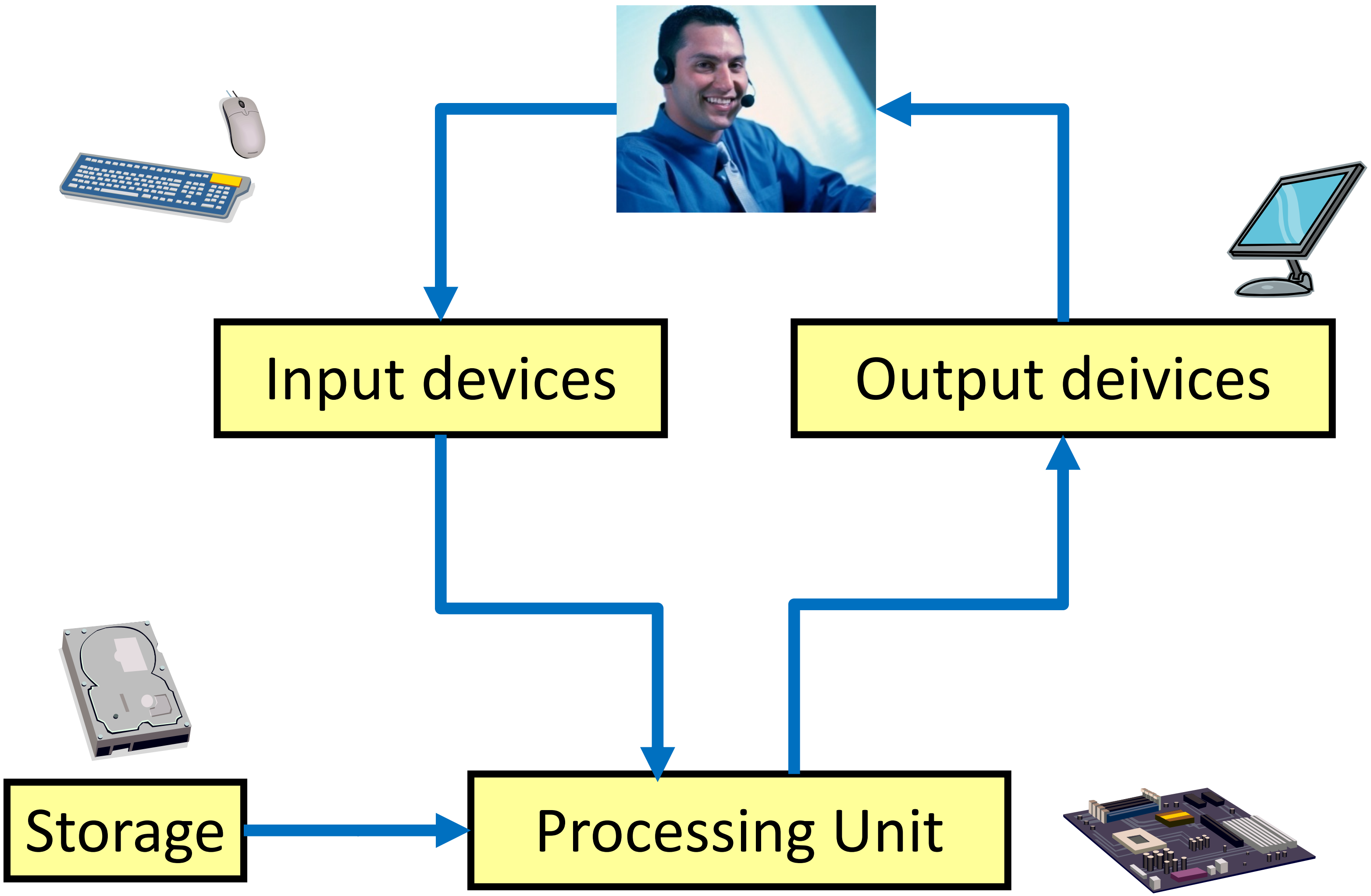
What we will do today

- Introduction
- Fundamental usage of KUE-CHIP2
- Problem 3.1
  - Trace the values in `ACC`, `PC`, `FLAG`, etc., while executing `ADD` and `ADC`.
- Problem 3.3 (1)
  - Check the clock frequency
  - Generate a single tone that are as much accurate 440 Hz as possible.
- Introduction of next problem

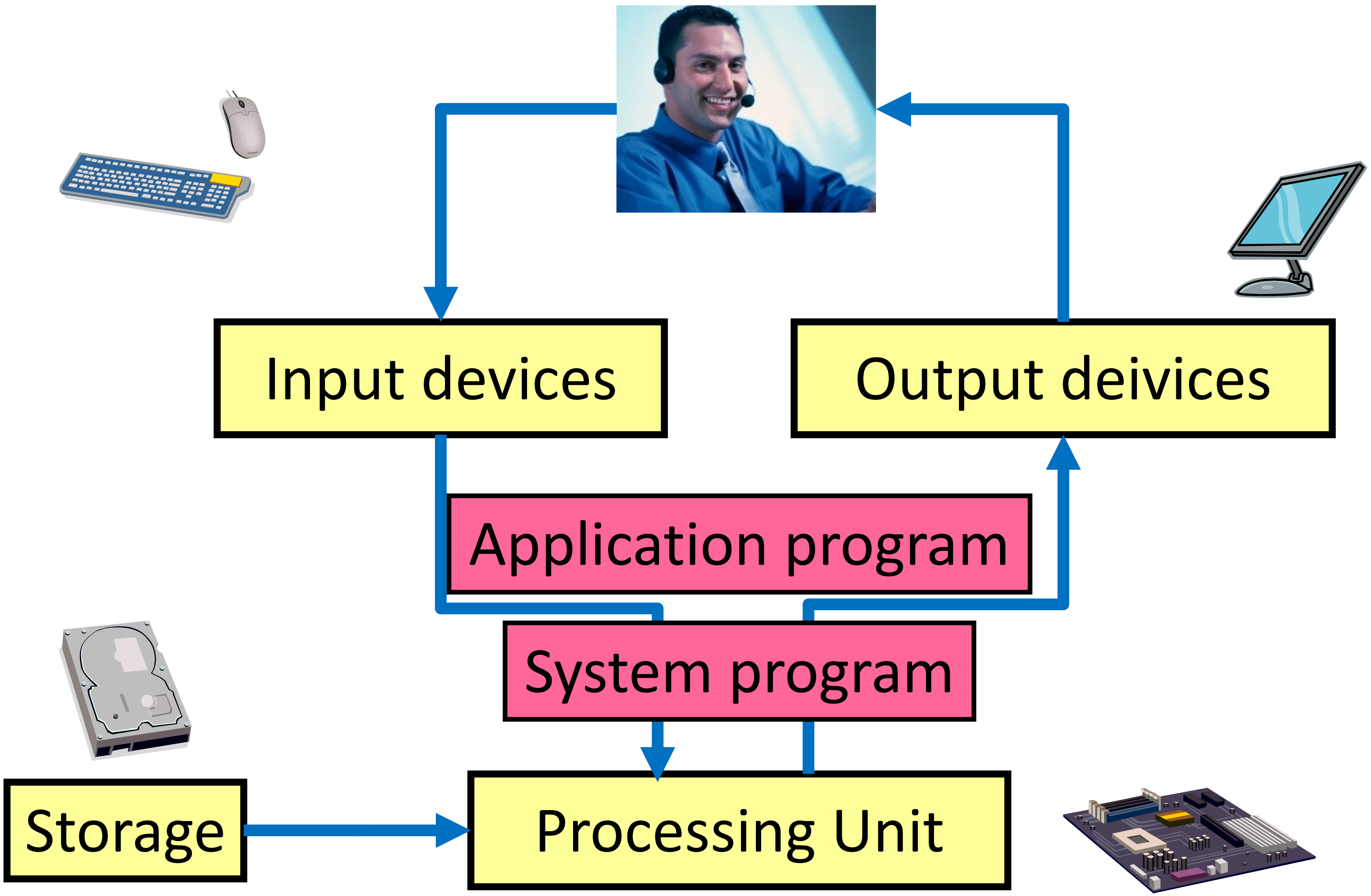
Relationships between a computer and a user



Hardware

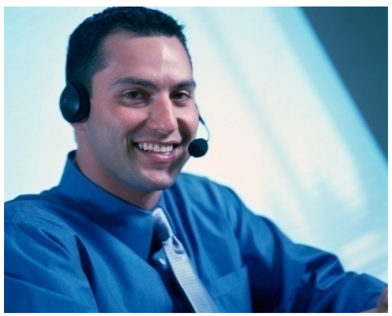


Software


Input devices
Output deivices
Application program
System program
Storage
Processing Unit

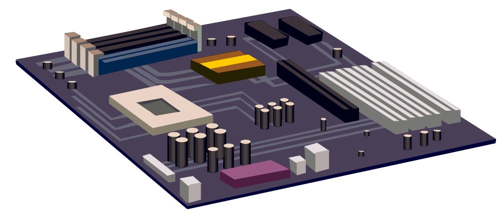
Question

- How does the processing unit understand programs (software)?



```
void swap(int v[], int k) {
  int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Program in
high-level language

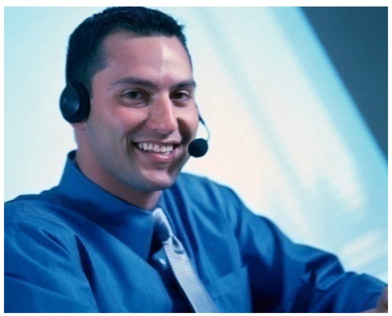


```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Program in
machine language

Answer (tentative)

- "Compiler" and "assembler" convert the problem



```
void swap(int v[], int k) {
  int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

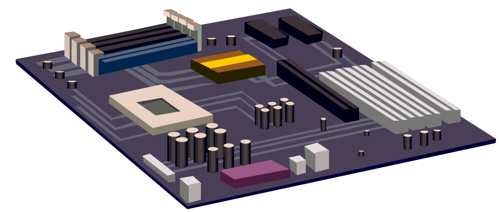
High-level language

compile

```
swap:
  muli $2, $5,4
  add  $2, $4,$2
  lw   $15, 0($2)
  lw   $16, 4($2)
  sw   $16, 0($2)
  sw   $15, 4($2)
  jr   $31
```

Assembly language

assemble

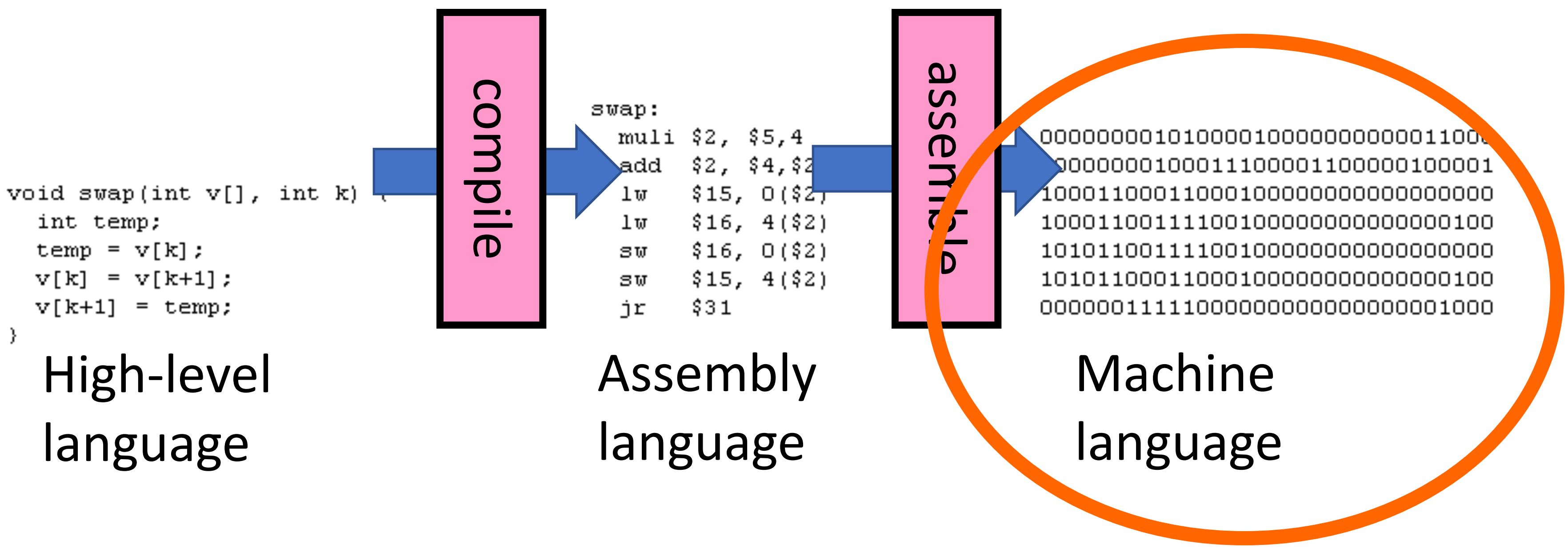


```
00000000101000010000000000011000
0000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Machine language

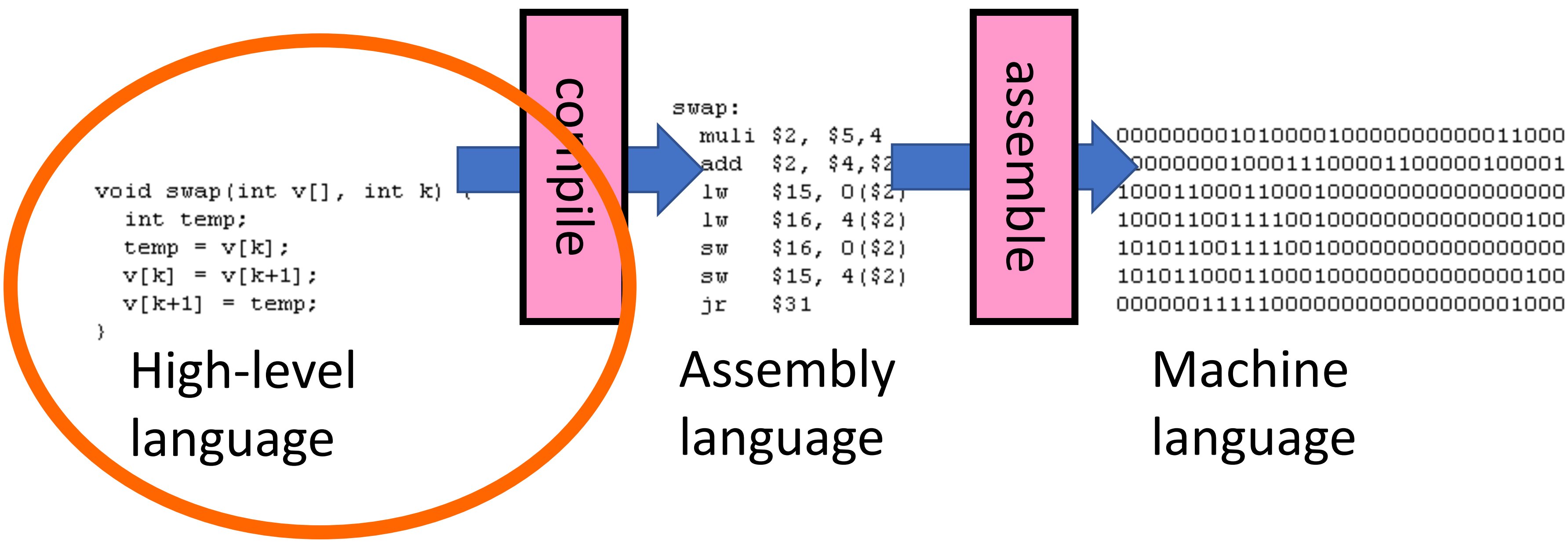
What is “machine language?”

- It can be understood and executed directly by CPU
- It is composed of 0 or 1
- It differs among CPUs



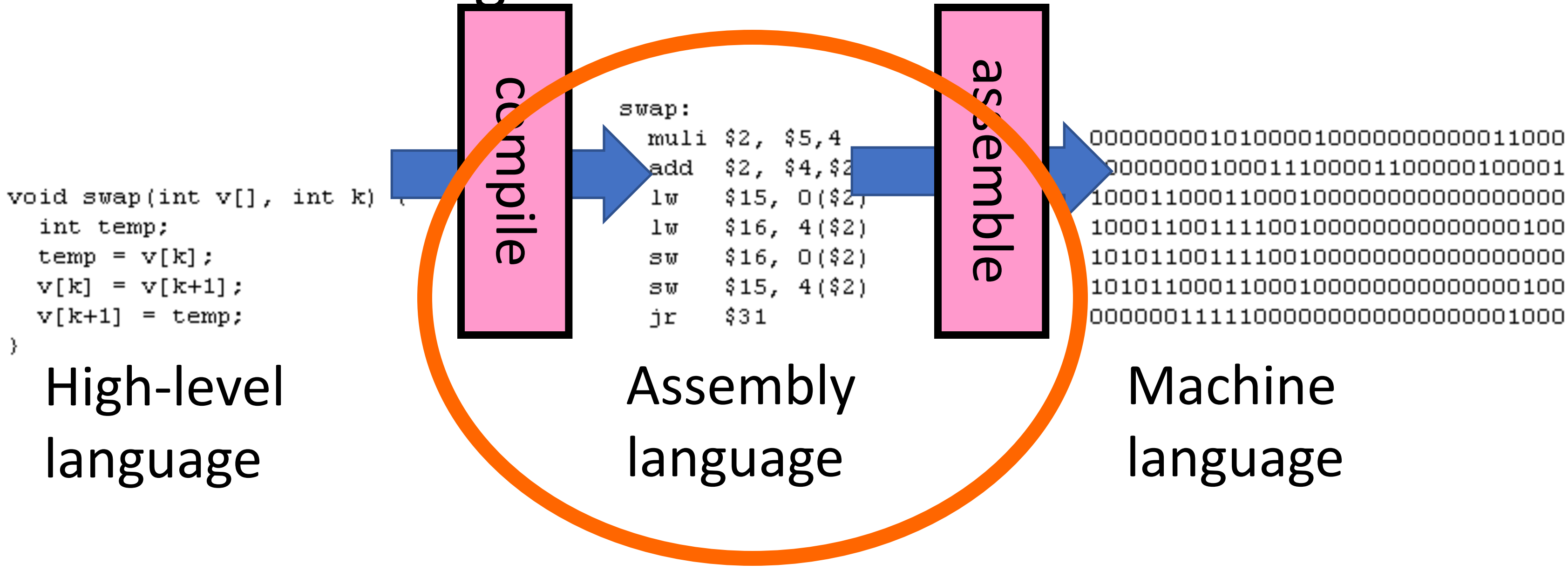
What is “high-level language?”

- It can be easily understood by human
- For e.g. C, C++, Java, Perl
- It is the same for all CPUs



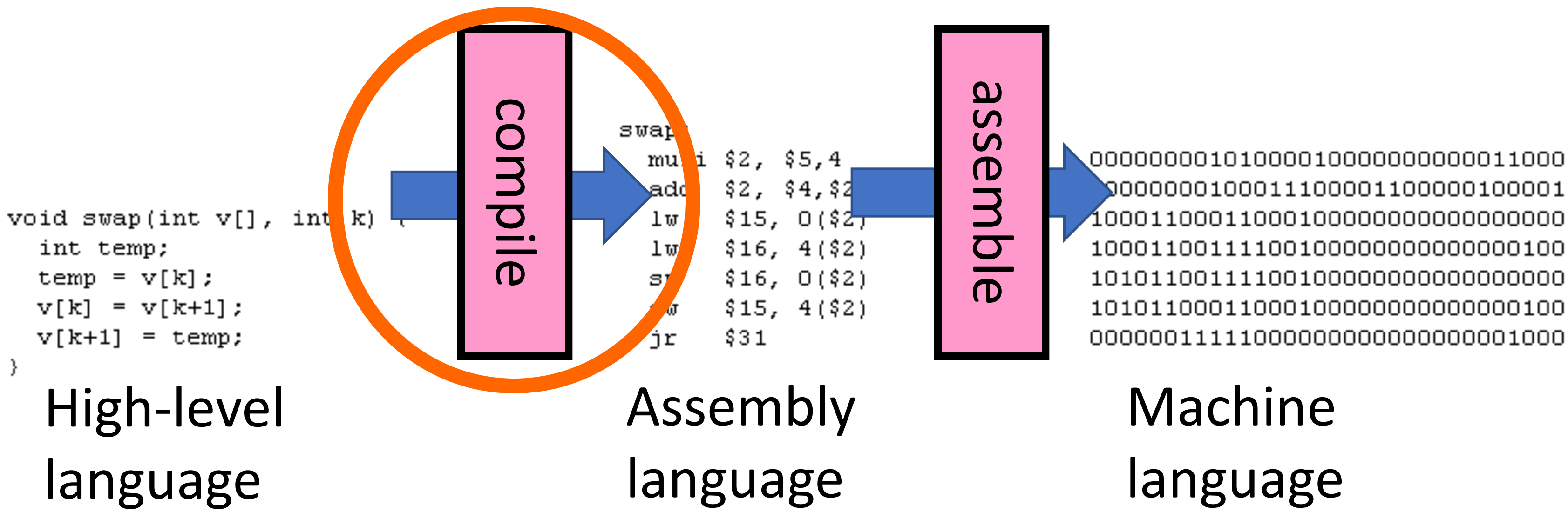
What is “Assembly language?”

- It is a translation of machine language
- It has a one-to-one correspondence with the machine language
- It differs among CPUs



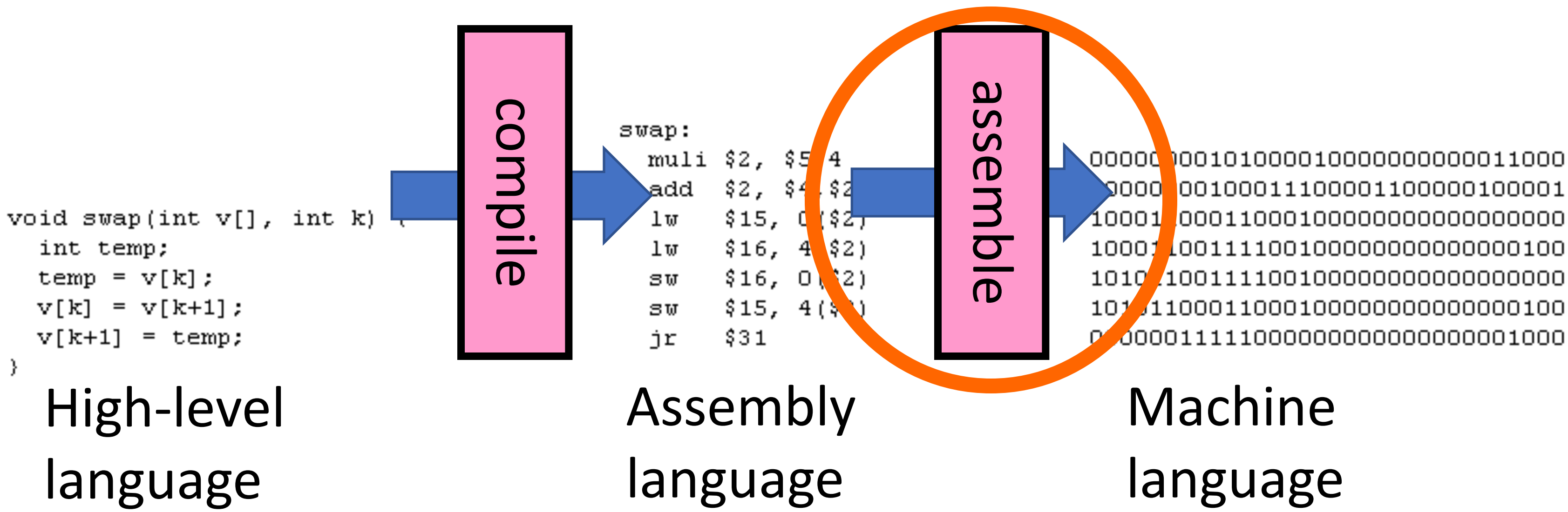
What is “compile?”

- It means to translate programs from a high-level language to an assembly language (or a machine language)



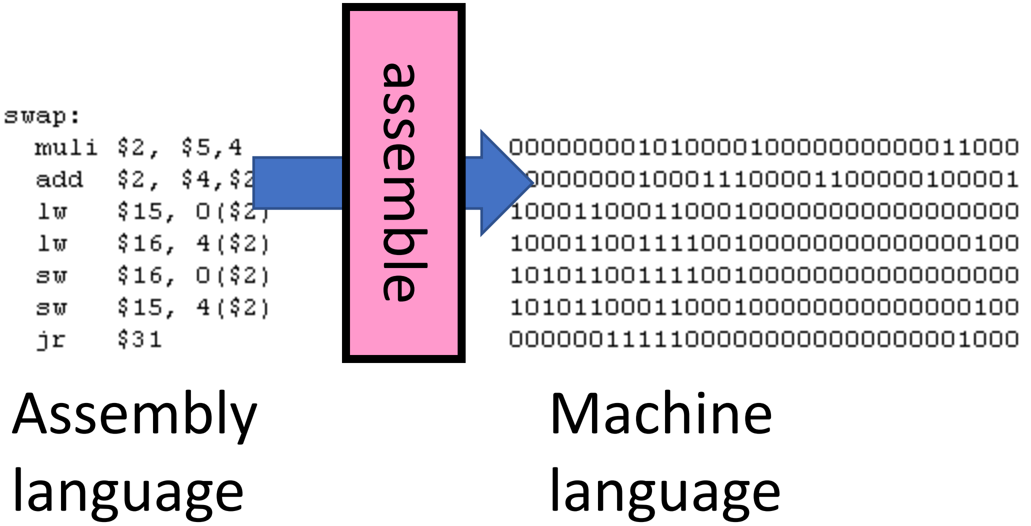
What is “assembly?”

- It is the process of translating programs from an assembly language to a machine language.



Flow of the theme

1. Programming in an assembly language
2. Manually assembling your own programs
3. Executing the programs and understanding the mechanisms.



Device used in this theme

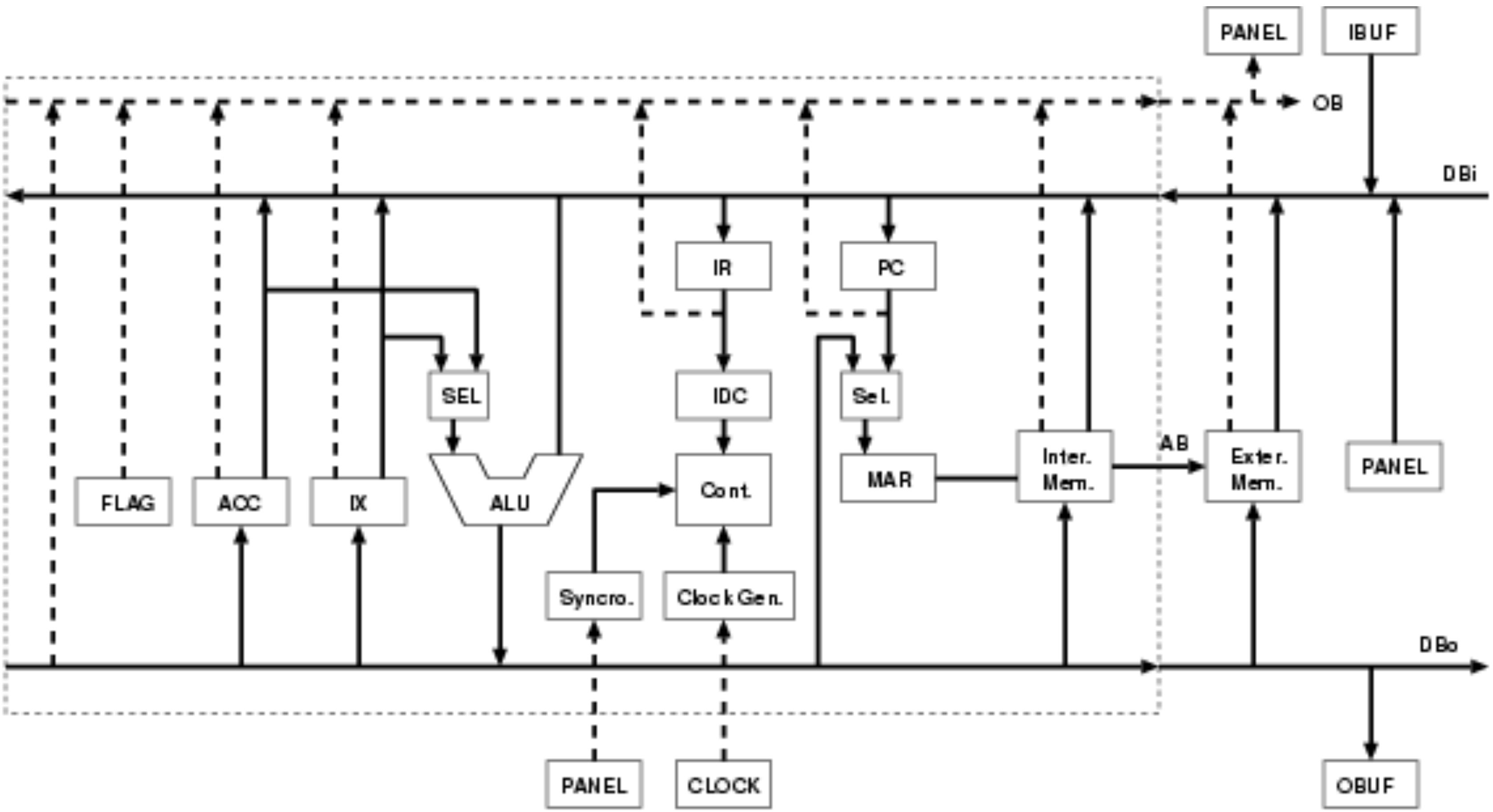
- KUE-CHIP2
- It is an educational 8-bit microprocessor
  = CPU

8 bits = 1 byte

0	0	0	1	0	0	1	1

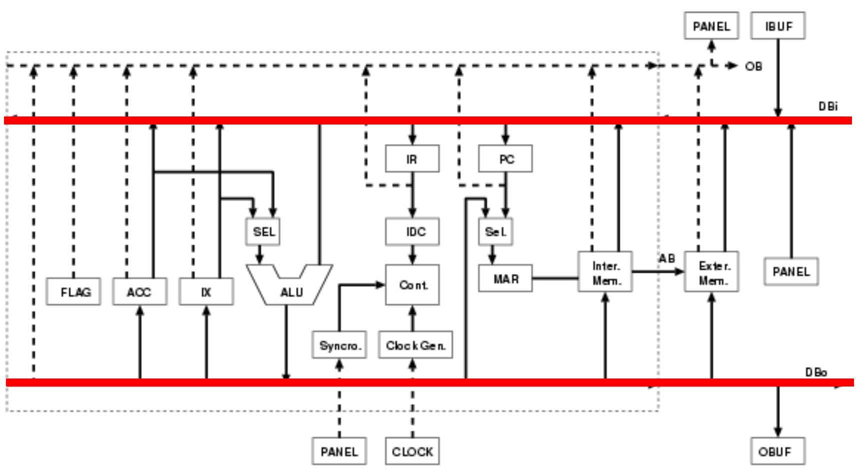
13h ← The "h" signifies that the number is in the hexadecimal notation (e.g. 13H,　0x13)

Structure of KUE-CHIP2 (p.22 Fig. 1)



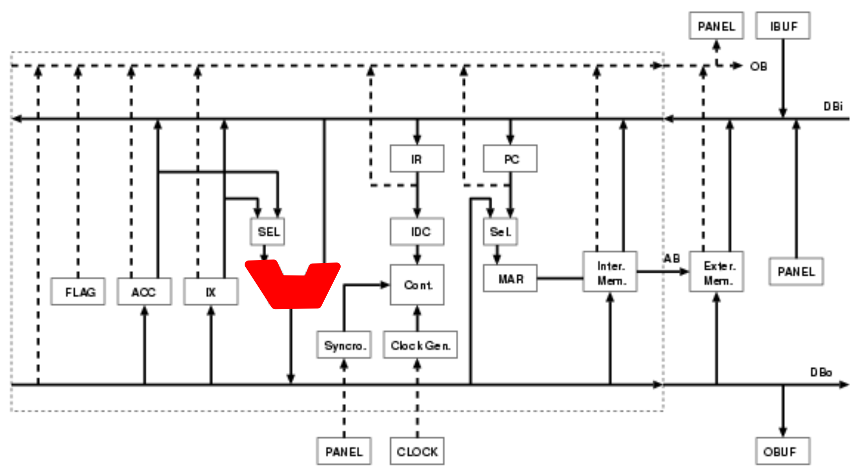
KUE-CHIP2: bus

- Input bus: It connects inputs and CPU
- Output bus: It connects outputs and CPU



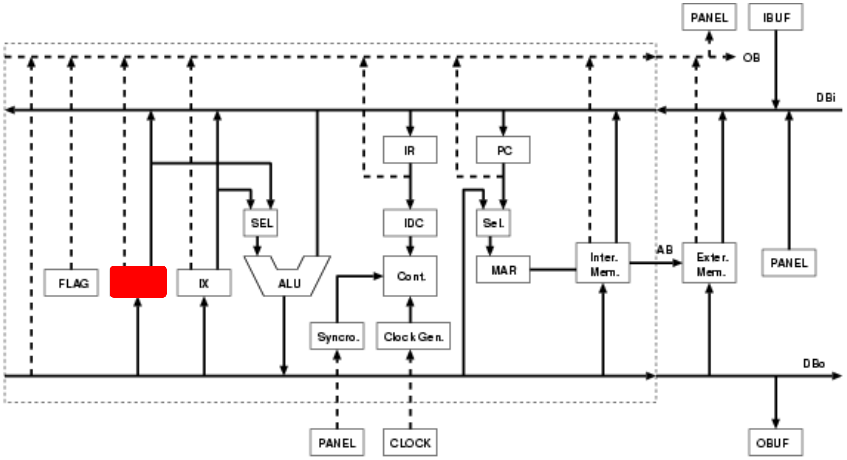
KUE-CHIP2: ALU

- ALU stands for “Arithmetic and Logic Unit” (演算ユニット)
- It performs arithmetic (算術) and logical (論理) operations and addresses calculation



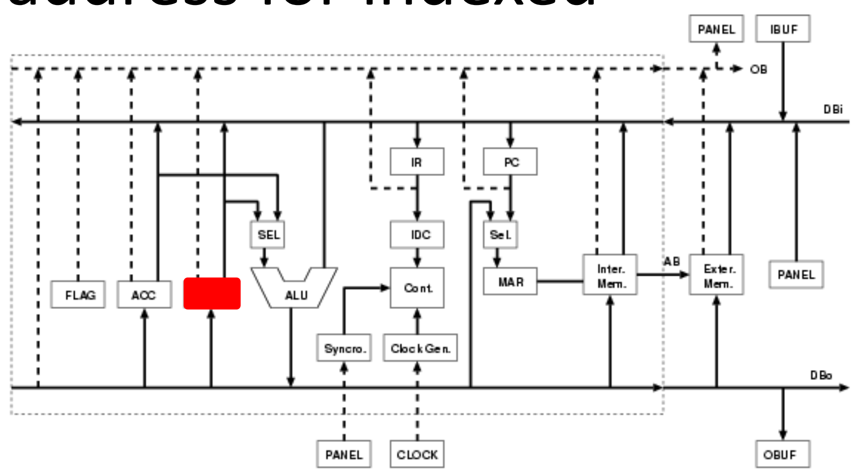
KUE-CHIP2: ACC

- ACC is the accumulator
- It is an 8-bit register for operations
- It stores operands and operated results



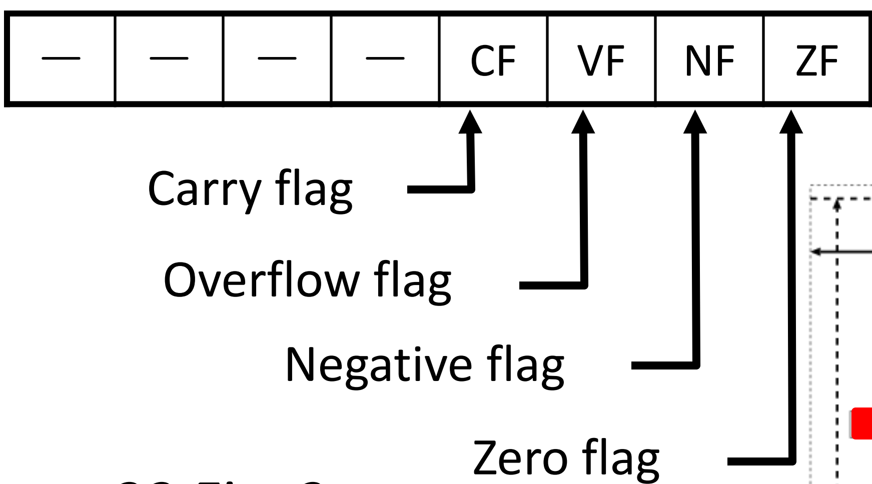
KUE-CHIP2: IX

- IX refers to index register
- It is an 8-bit register used for operations
- It stores operands and operated results
- It is used for indexing an address for indexed address (修飾アドレス)

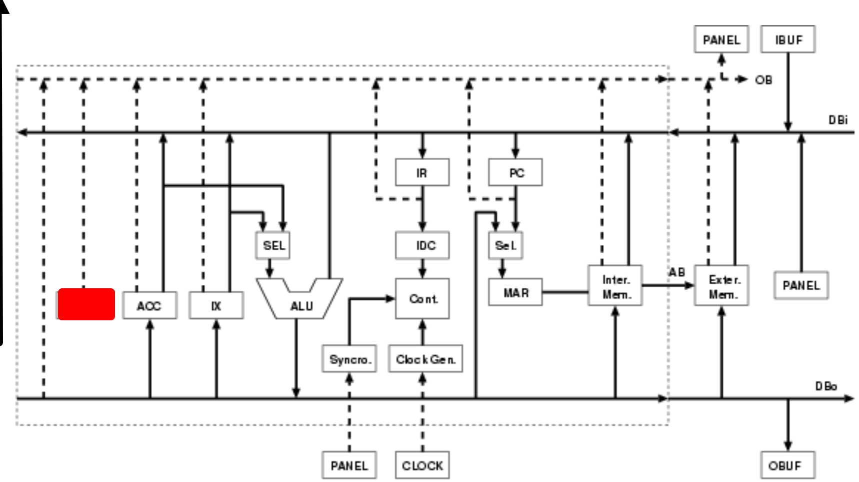


KUE-CHIP2: FLAG

- Flag register
- It is changed by operation results

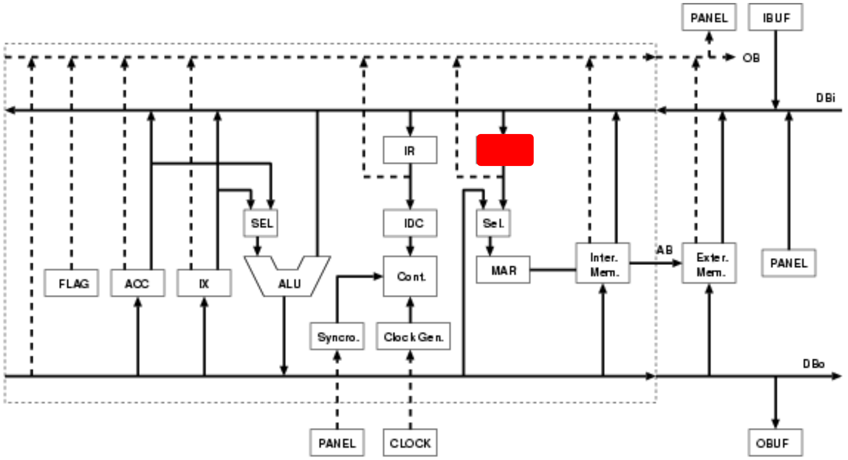


p.22 Fig. 2



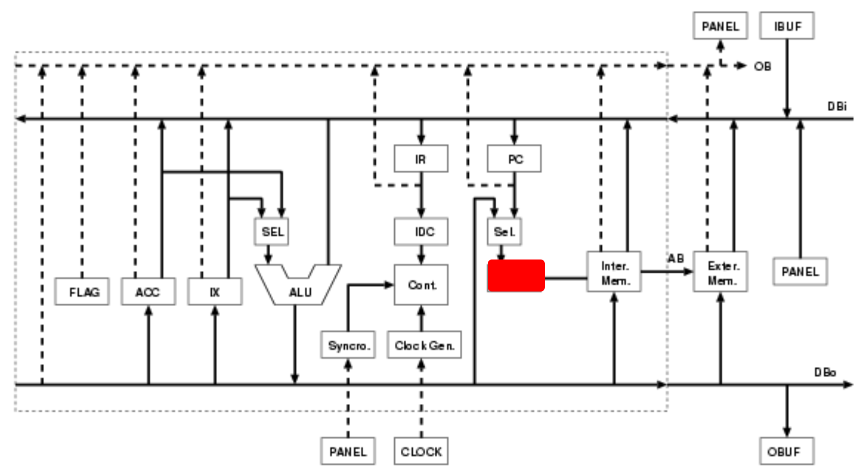
KUE-CHIP2: PC

- PC refers to program counter
- It stores the 8-bit address on the memory of the subsequent command.



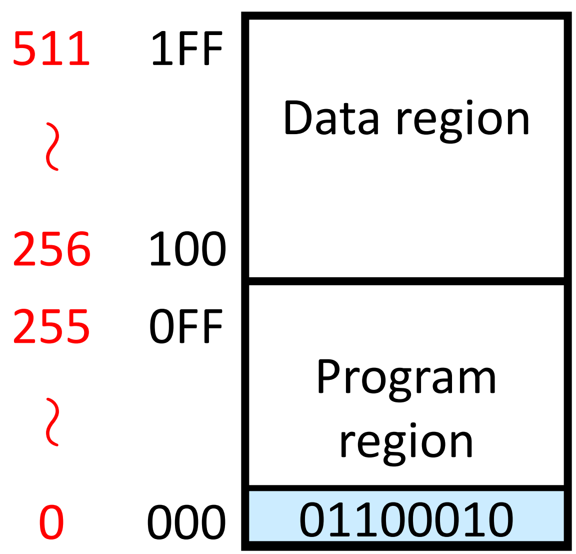
KUE-CHIP2: MAR

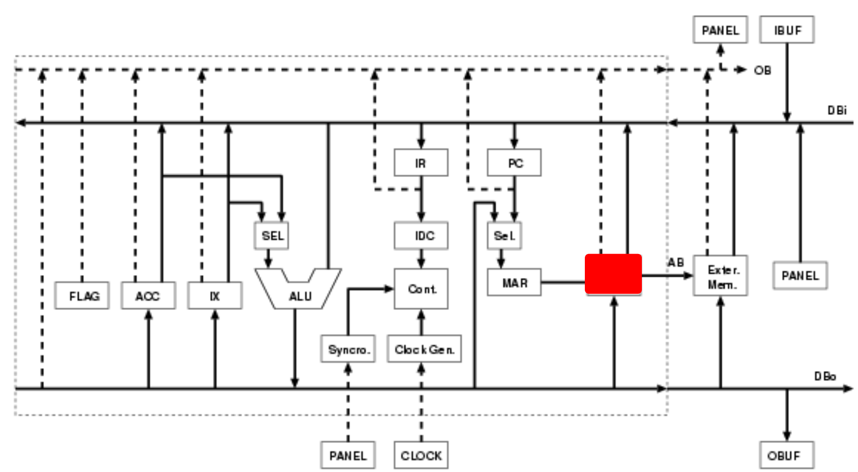
- MAR is the "memory address register" (8 bits)
- It stores the memory address from which data will be fetched or to which data will be sent



KUE-CHIP2: Internal memory (内部メモリ)

- It consists of 512 bytes. The indexing unit is byte
- Program region: 0-255 addresses
- Data region: 256-511 addresses





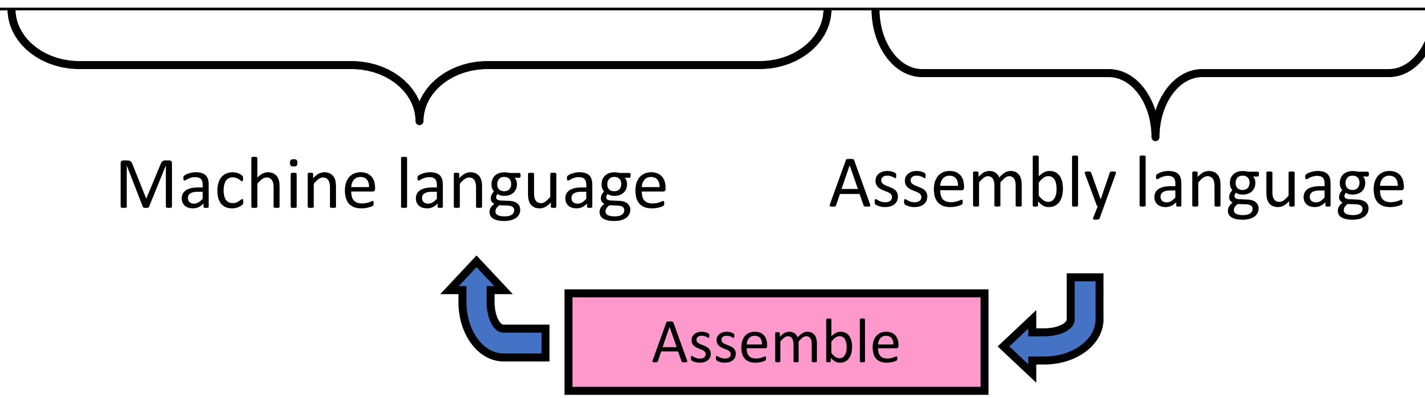
p.23 Fig. 3

Assembly language for KUE-CHIP2

- Commands: p.24 Table 1
- Language specification: pp.35-38 Appendix A
- Format for machine language: 1 or 2 byte (p.23 Fig. 4)

Example (p.30, List 2)

address	data				command	operands
00:	**0110**	**001-**	**0000**	**0001**	**LD**	**ACC, 01h**
02:	**0001**	**0---**			**OUT**	
03:	**0100**	**0111**			**RLL**	**ACC**
04:	**0011**	**0000**	**0000**	**0010**	**BA**	**02h**



Example (p.30, List 2)

address	data				command	operands
00:	**0110**	**001-**	**0000**	**0001**	**LD**	**ACC, 01h**
02:	**0001**	**0---**			**OUT**	
03:	**0100**	**0111**			**RLL**	**ACC**
04:	**0011**	**0000**	**0000**	**0010**	**BA**	**02h**

Load the value "01" in the ACC

Example (p.30, List 2)

address	data				command	operands
00:	**0110**	**001-**	**0000**	**0001**	**LD**	**ACC, 01h**
02:	**0001**	**0---**			**OUT**	
03:	**0100**	**0111**			**RLL**	**ACC**
04:	**0011**	**0000**	**0000**	**0010**	**BA**	**02h**

Output the content of ACC to the output buffer (OBUF)

Example (p.30, List 2)

address	data				command	operands
00:	0110	001-	0000	0001	LD	ACC, 01h
02:	0001	0---			OUT	
03:	0100	0111			**RLL**	**ACC**
04:	0011	0000	0000	0010	BA	02h

Logically left rotate (論理左回転) the content of ACC, and store the rotated result

0 0 0 0 0 0 0 1

0 0 0 0 0 0 1 0

Example (p.30, List 2)

address	data				command	operands
00:	**0110**	**001-**	**0000**	**0001**	**LD**	**ACC, 01h**
02:	**0001**	**0---**			**OUT**	
03:	**0100**	**0111**			**RLL**	**ACC**
04:	**0011**	**0000**	**0000**	**0010**	**BA**	**02h**

Always return the "02" address

How to assemble (1/4)

- Command table (p.37, Table 8)
- Assembly "**LD ACC,01h**"

0	1	1	0	0	0	1	-	0	0	0	0	0	0	0	1

Rsm	0	1	0	0	A	1	s	m	×	Rotate sm
LD	0	1	1	0	A		B		○	LoaD
ST	0	1	1	1	A		B		◎	STore
SBC	1	0	0	0	A		B		○	SuB with Carry

How to assembly (1/4)

- Command table (p.37, Table 8)
- Assembly "**LD ACC,01h**"

0	1	1	0	0	0	1	-	0	0	0	0	0	0	0	1

A | B

The value in operands

A = 0:ACC
A = 1:IX

B = 000:ACC
B = 001:IX
B = 01-:Immediate (即値)
B = 100:Direct (直接)(P)
B = 101:Direct(D)
B = 110:Indexed (修飾)(P)
B = 111:Indexed(D)

How to assembly (1/4)

- Command table (p.37, Table 8)
- Assembly "**LD ACC,01h**"

0	1	1	0	0	0	1	-	0	0	0	0	0	0	0	1

How to assembly (2/4)

- Command table (p.37, Table 8)
- Assembly "**OUT**"

0	0	0	1	0	-	-	-

	0	1	0	1	-	-	-	-	×	
OUT	0	0	0	1	0	-	-	-	×	**OUTput**
IN	0	0	0	1	1	-	-	-	×	**INput**
RCF	0	0	1	0	0	-	-	-	×	**Reset CF**

How to assembly (3/4)

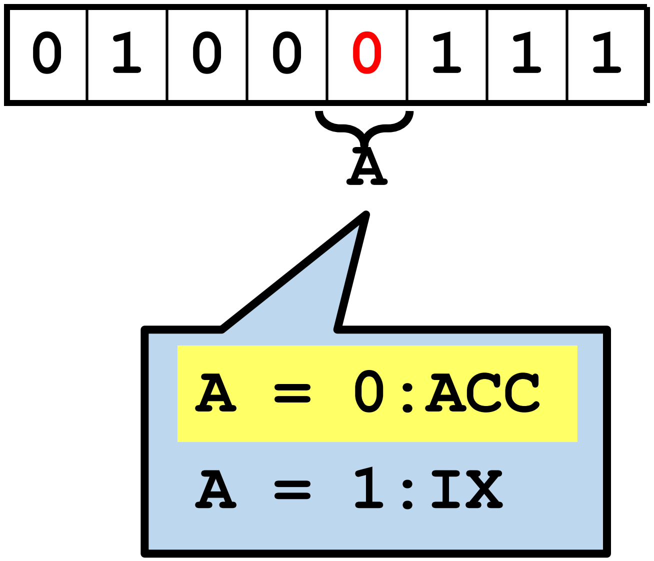
- Command table (p.37, Table 8)
- Assembly "**RLL ACC**"

0	1	0	0	0	1	1	1

Rsm	**0**	**1**	**0**	**0**	**A**	**1**	**s**	**m**	✕	**Rotate sm**
LD	**0**	**1**	**1**	**0**	**A**		**B**		○	**LoaD**
ST	**0**	**1**	**1**	**1**	**A**		**B**		◎	**STore**
SBC	**1**	**0**	**0**	**0**	**A**		**B**		○	**SuB with Carry**

How to assembly (3/4)

- Command table (p.37, Table 8)
- Assembly “**RLL ACC**”



How to assembly (3/4)

- Command table (p.37, Table 8)
- Assembly “**RLL ACC**”

0	1	0	0	0	1	1	1

s m

RA	0	0	**Right Arithmetically**
LA	0	1	**Left Arithmetically**
RL	1	0	**Right Logically**
LL	1	1	**Left Logically**

How to assembly (4/4)

- Command table (p.37, Table 8)
- Assembly "**BA 02h**"

0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0

RCF	0	0	1	0	0	-	-	-	×	Reset CF
SCF	0	0	1	0	1	-	-	-	×	Set CF
Bcc	0	0	1	1		c	c		◎	Branch cc
Ssm	0	1	0	0	A	0	s	m	×	Shift sm

How to assembly (4/4)

- Command table (p.37, Table 8)
- Assembly “**BA** **02h**”

0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0

c c

A	0	0	0	0	**Always**
VF	1	0	0	0	**on oVerFlow**
NZ	0	0	0	1	**on Not Zero**
Z	1	0	0	1	**on Zero**

Example (p.30, List 2)

address	data				command	operands
00:	**0110**	**001-**	**0000**	**0001**	**LD**	**ACC, 01h**
02:	**0001**	**0---**			**OUT**	
03:	**0100**	**0111**			**RLL**	**ACC**
04:	**0011**	**0000**	**0000**	**0010**	**BA**	**02h**

"–" represents "`do not care`."
It can be replaced with either "`0`" or "`1`."

Example (p.30, List 2)

address	data	command	operands
00:	0110 0010 01: 0000 0001	LD	ACC, 01h
02:	0001 0000	OUT	
03:	0100 0111	RLL	ACC
04:	0011 0000 05: 0000 0010	BA	02h

Finish to assemble

What we will do today

- Introduction
- Fundamental usage of KUE-CHIP2
- Problem 3.1
  - Trace the values in `ACC`, `PC`, `FLAG`, etc., while executing `ADD` and `ADC`.
- Problem 3.3 (1)
  - Check the clock frequency
  - Generate a single tone that are as much accurate 440 Hz as possible.
- Introduction of next problem

Execution of programs

- Follow Sec. 2.5 (pp.26--32)
- Caution:
  - Plug in after the board is connected to the adapter.
  - Plug into the outlets fixed on the desks
  - Do not touch the condenser beside the power switch
  - Press the RESET button before execution
- After all of you have finished the procedure, we will proceed to the next step.

Supplementary explanation for operating the board

- The SS switch executes, stops, and resumes programs.
- The CLKFRQ dial changes the speed of execution.
- The SEL switch displays the contents of ACC, PC, FLAG, MAR, etc.
- The SI switch executes a command (the step execution)

What we will do today

- Introduction
- Fundamental usage of KUE-CHIP2
- Problem 3.1
  - Trace the values in `ACC`, `PC`, `FLAG`, etc., while executing `ADD` and `ADC`.
- Problem 3.3 (1)
  - Check the clock frequency
  - Generate a single tone that are as much accurate 440 Hz as possible.
- Introduction of next problem

How does a command execute?

- A command executes by clock
- A single cycle of the clock corresponds to a single execution phase.

- Each command of KUE-CHIP2 consumes 3—5 phases.
  - P0, P1: common in all commands
  - After P2: differs among the commands

p.25, Table 2

Example for trace of the execution (p.26, List 1)

address	data		label	command	operands
			D1:	**EQU**	**80h**
			D2:	**EQU**	**81h**
			ANS:	**EQU**	**82h**
00:	**64**	**80**		**LD**	**ACC,[D1]**
02:	**B4**	**81**		**ADD**	**ACC,[D2]**
04:	**74**	**82**		**ST**	**ACC,[ANS]**
06:	**0F**			**HLT**	
				END	
80:	**03**				
81:	**FD**				

”D1” corresponds to “80h”.
(Similar to variable declaration or initialization.)

Example for trace of the execution (p.26, List 1)

address	data		label	command	operands
			D1:	**EQU**	**80h**
			D2:	**EQU**	**81h**
			ANS:	**EQU**	**82h**
00:	**64**	**80**		**LD**	**ACC,[D1]**
02:	**B4**	**81**		**ADD**	**ACC,[D2]**
04:	**74**	**82**		**ST**	**ACC,[ANS]**
06:	**0F**			**HLT**	
				END	
80:	**03**				
81:	**FD**				

It transfers the contents of memory location `D1` to `ACC`

Example for trace of the execution (p.26, List 1)

address	data		label	command	operands
			D1:	EQU	80h
			D2:	EQU	81h
			ANS:	EQU	82h
00:	64	80		LD	ACC,[D1]
02:	B4	81		ADD	ACC,[D2]
04:	74	82		ST	ACC,[ANS]
06:	0F			HLT	
				END	
80:	03				
81:	FD				

It adds the contents of ACC and the D2 address in the program region

Example for trace of the execution (p.26, List 1)

address	data		label	command	operands
			D1:	**EQU**	**80h**
			D2:	**EQU**	**81h**
			ANS:	**EQU**	**82h**
00:	**64**	**80**		**LD**	**ACC,[D1]**
02:	**B4**	**81**		**ADD**	**ACC,[D2]**
04:	**74**	**82**		**ST**	**ACC,[ANS]**
06:	**0F**			**HLT**	
				END	
80:	**03**				
81:	**FD**				

It stores the content of `ACC` to the `ANS` address in the program region

Example for trace of the execution (p.26, List 1)

address	data		label	command	operands
			D1:	**EQU**	**80h**
			D2:	**EQU**	**81h**
			ANS:	**EQU**	**82h**
00:	**64**	**80**		**LD**	**ACC,[D1]**
02:	**B4**	**81**		**ADD**	**ACC,[D2]**
04:	**74**	**82**		**ST**	**ACC,[ANS]**
06:	**0F**			**HLT**	
				END	
80:	**03**				
81:	**FD**				

It halts the execution of the program

Example for trace of the execution (p.26, List 1)

address	data		label	command	operands
			D1:	EQU	80h
			D2:	EQU	81h
			ANS:	EQU	82h
00:	64	80		LD	ACC,[D1]
02:	B4	81		ADD	ACC,[D2]
04:	74	82		ST	ACC,[ANS]
06:	0F			HLT	
				END	

80:	03
81:	FD

Input "03" in the 80th address and "FD (−3)" in the 81st address

Example for trace of the execution (p.26, List 1)

address	data		label	command	operands
			D1:	**EQU**	**80h**
			D2:	**EQU**	**81h**
			ANS:	**EQU**	**82h**
00:	**64**	**80**		**LD**	**ACC,[D1]**
02:	**B4**	**81**		**ADD**	**ACC,[D2]**
04:	**74**	**82**		**ST**	**ACC,[ANS]**
06:	**0F**			**HLT**	
				END	
80:	**03**				
81:	**FD**				

This is the assembled program in the hexadecimal notation

Trace of the execution

LD ACC,[D1]

ACC → **A**, [D1] → **B**

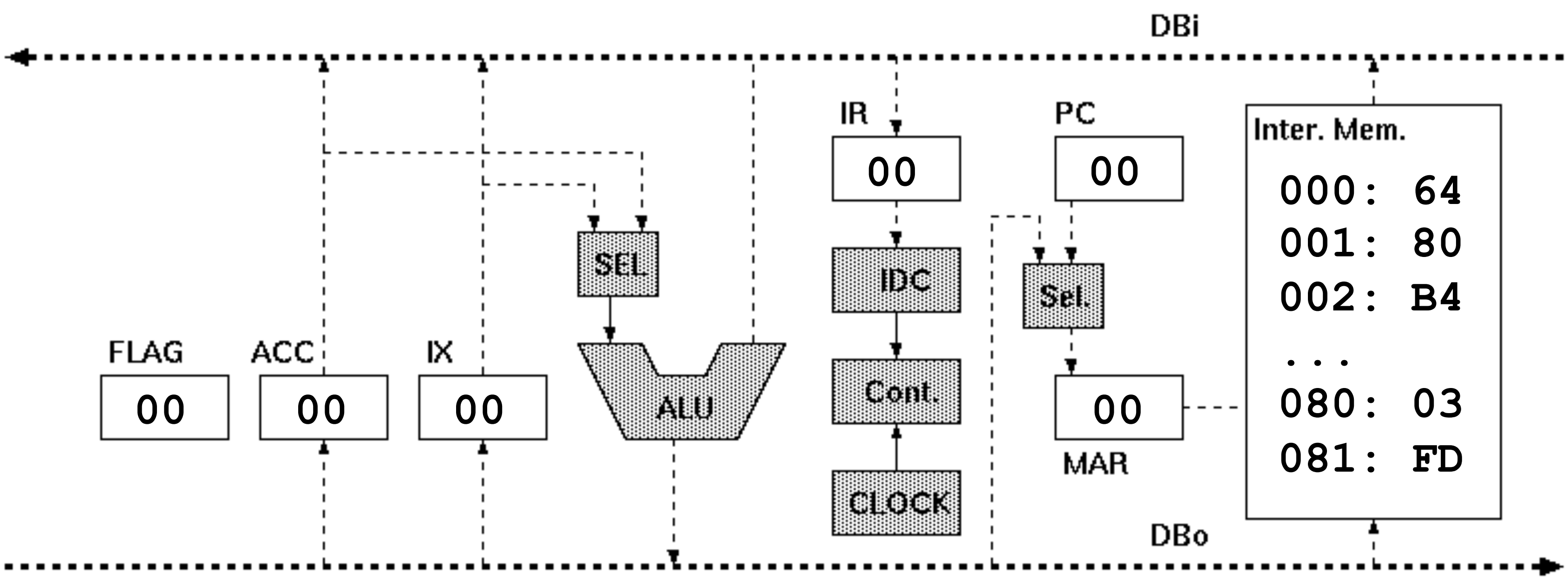
p.25 Table 2

According to type of B,
the procedure is changed.

<table>
<tr><th colspan="2"></th><th>P0</th><th>P1</th><th>P2</th><th>P3</th><th>P4</th></tr>
<tr><td rowspan="3">LD</td><td>ACC
IX</td><td rowspan="3">(PC)→MAR
PC++</td><td rowspan="3">(Mem)→IR</td><td>(A)→B</td><td></td><td></td></tr>
<tr><td>d</td><td rowspan="2">(PC)→MAR
PC++</td><td>(Mem)→A</td><td></td></tr>
<tr><td>[d]
(d)</td><td>(Mem)→MAR</td><td>(Mem)→A</td></tr>
</table>

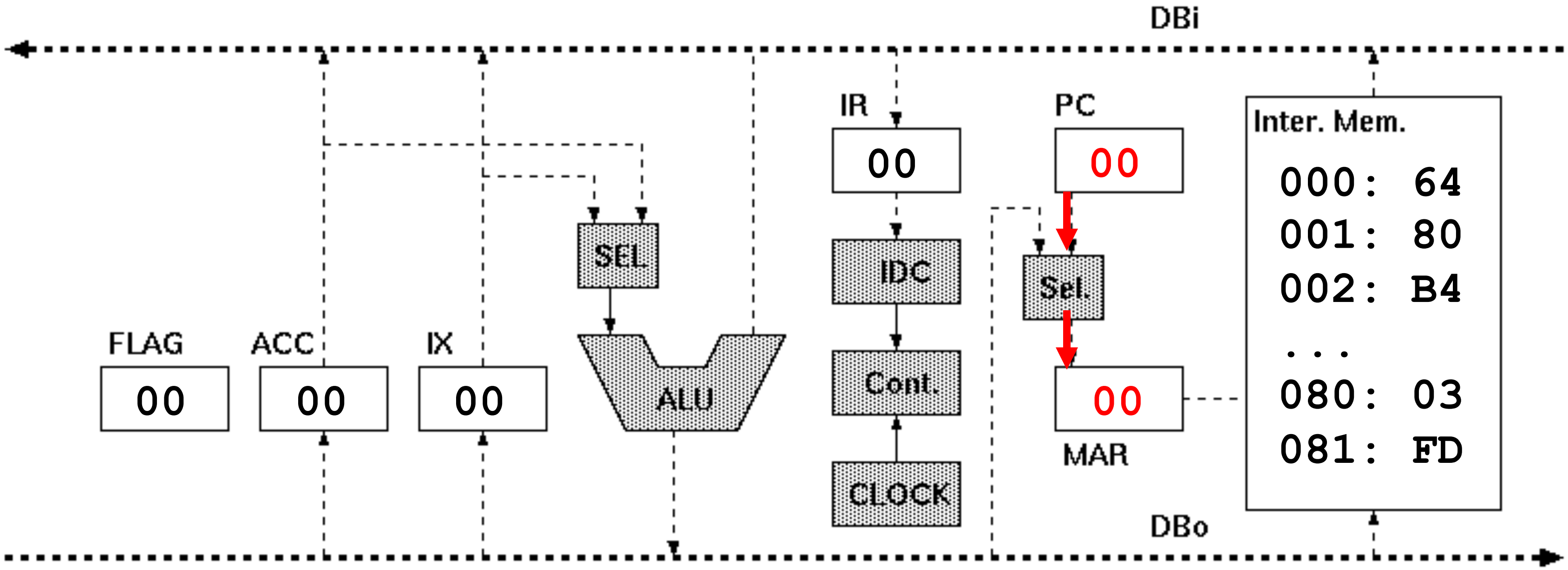
Trace of the execution

LD ACC,[D1]



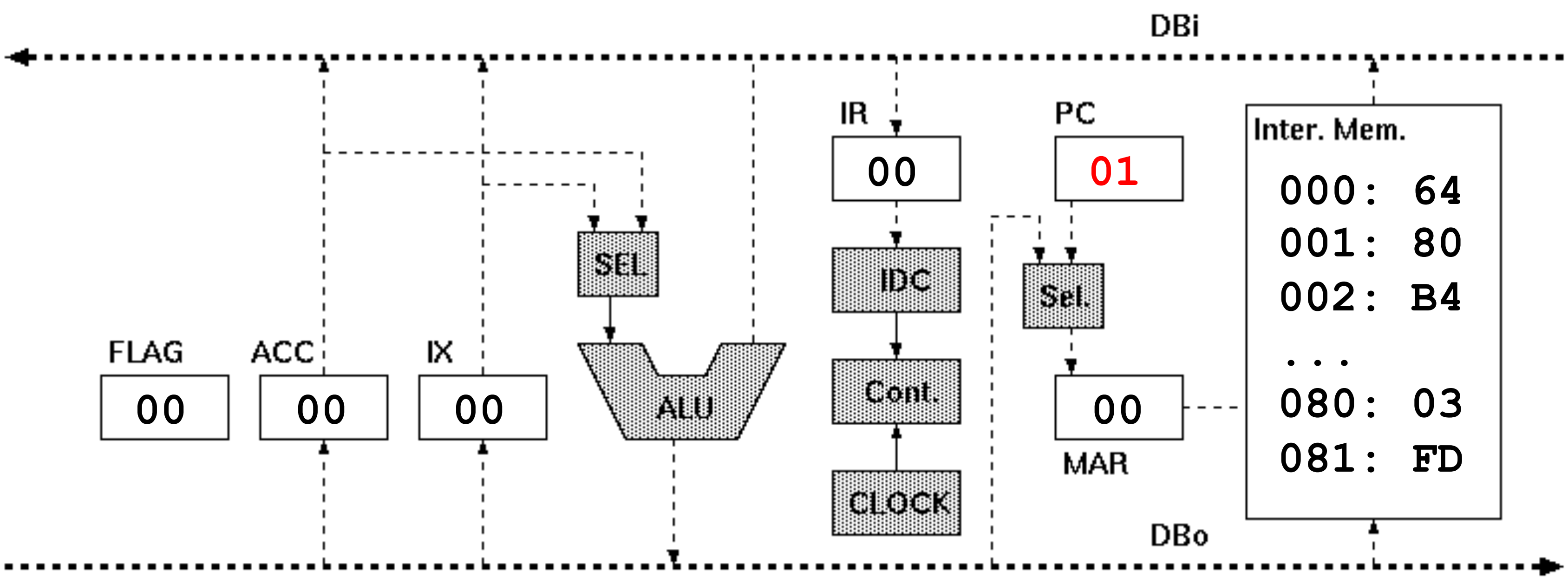
Trace of the execution

LD ACC,[D1]   P0: (PC)→MAR, PC++



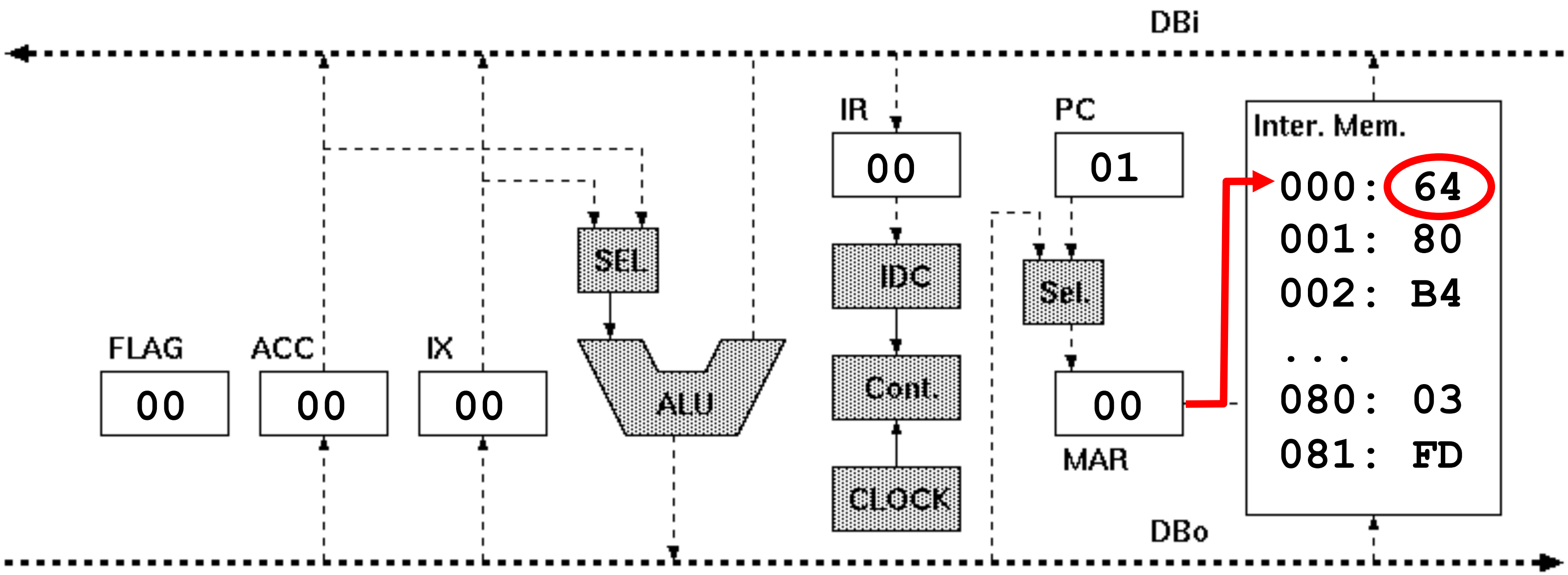
Trace of the execution

LD ACC,[D1] **P0: (PC)→MAR, PC++**



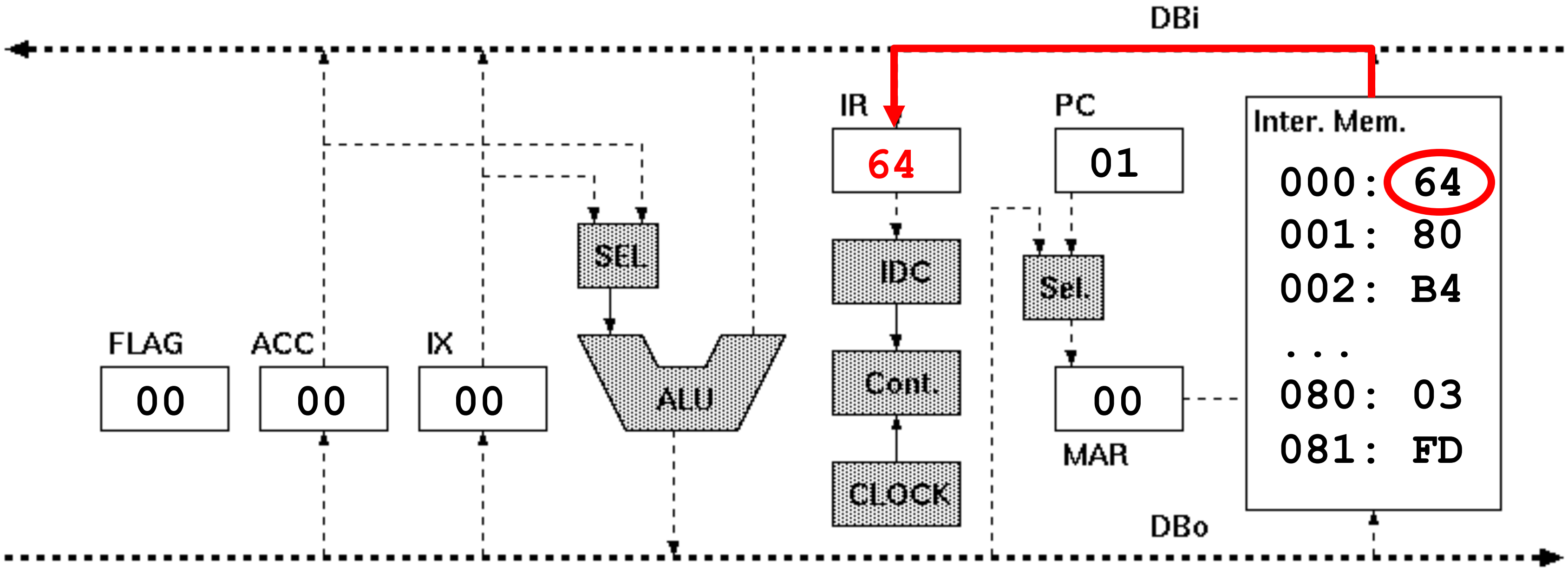
Trace of the execution

LD ACC,[D1] **P1: (Mem)→IR**



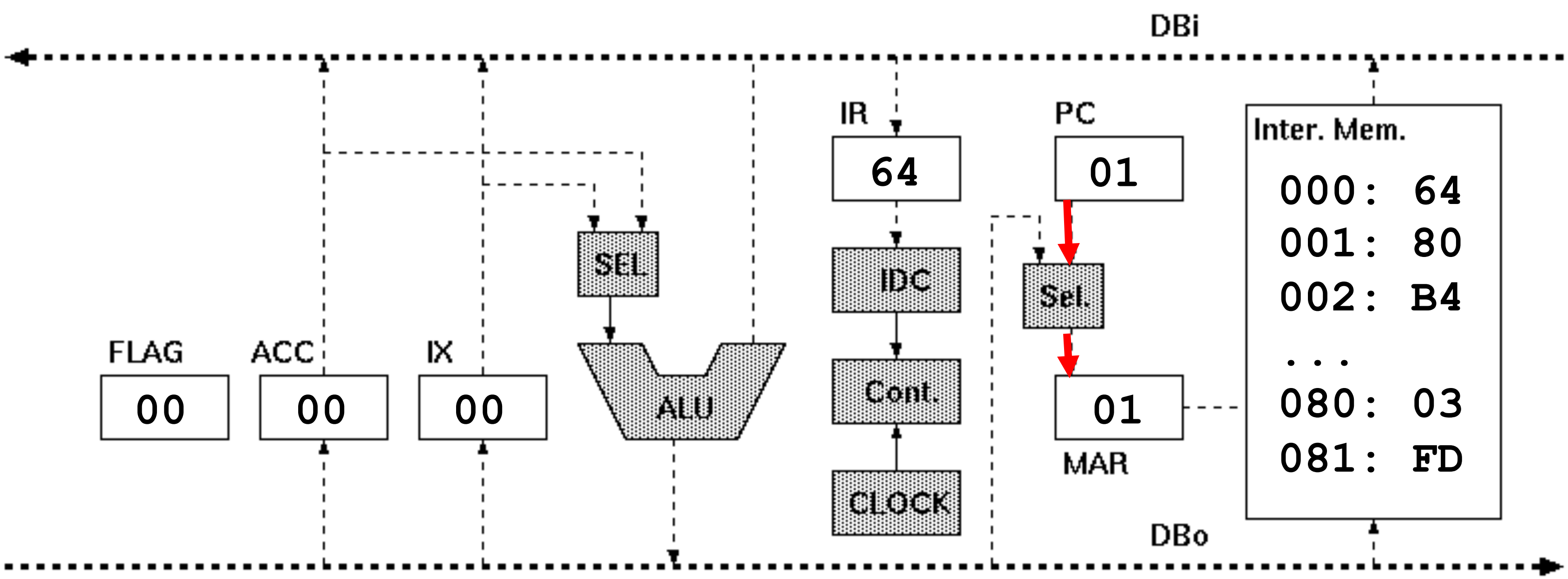
Trace of the execution

LD ACC,[D1] **P1: (Mem)→IR**



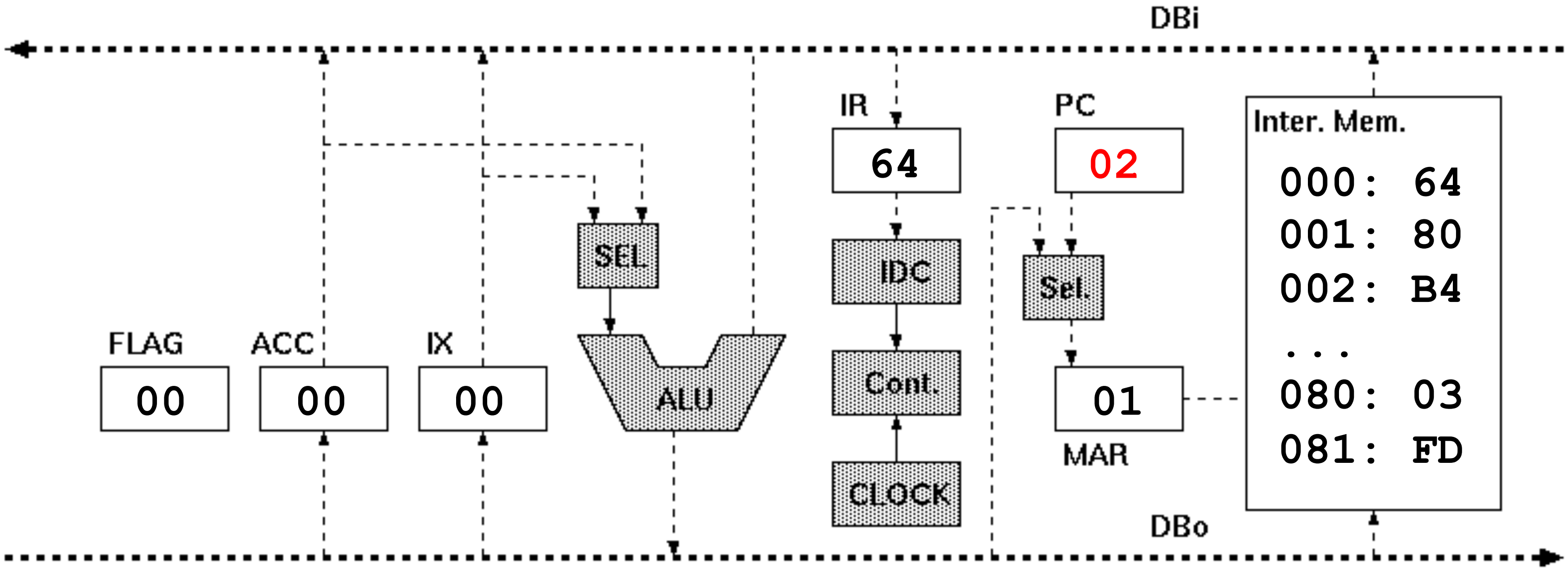
Trace of the execution

LD ACC,[D1] P2: (PC)→MAR, PC++



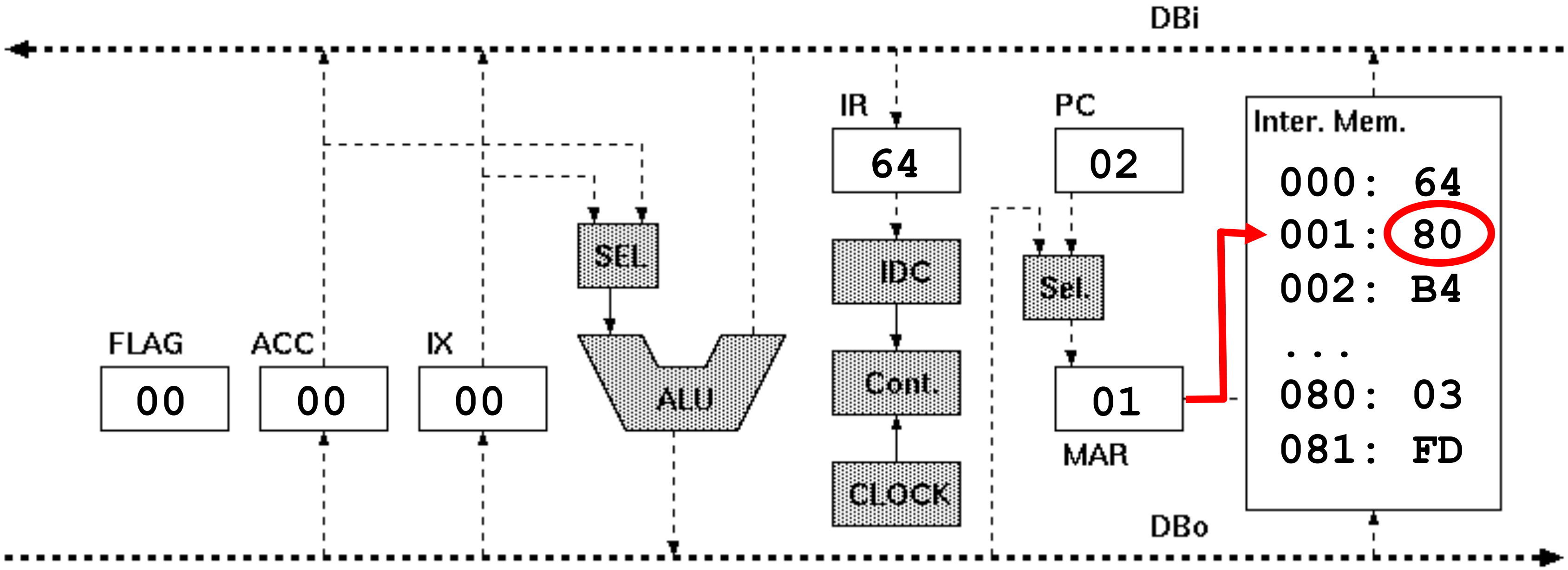
Trace of the execution

LD ACC,[D1] **P2: (PC)→MAR, PC++**



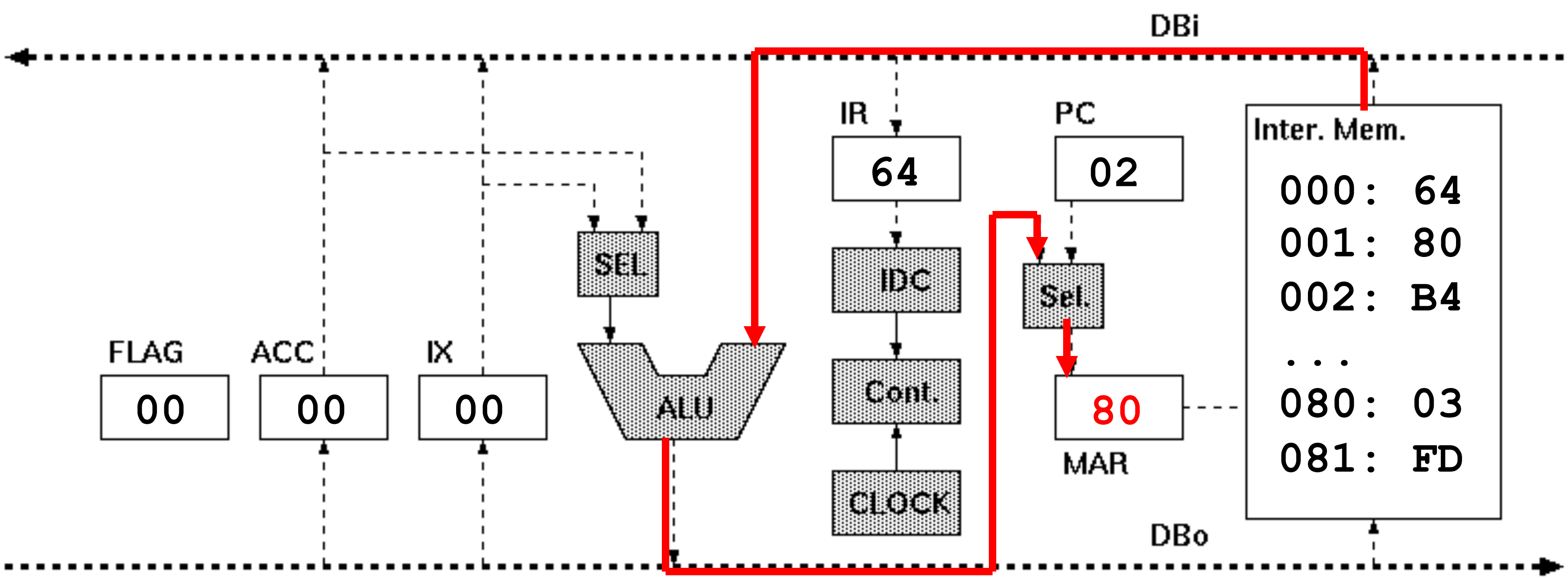
Trace of the execution

LD ACC,[D1] **P3: (Mem)→MAR**



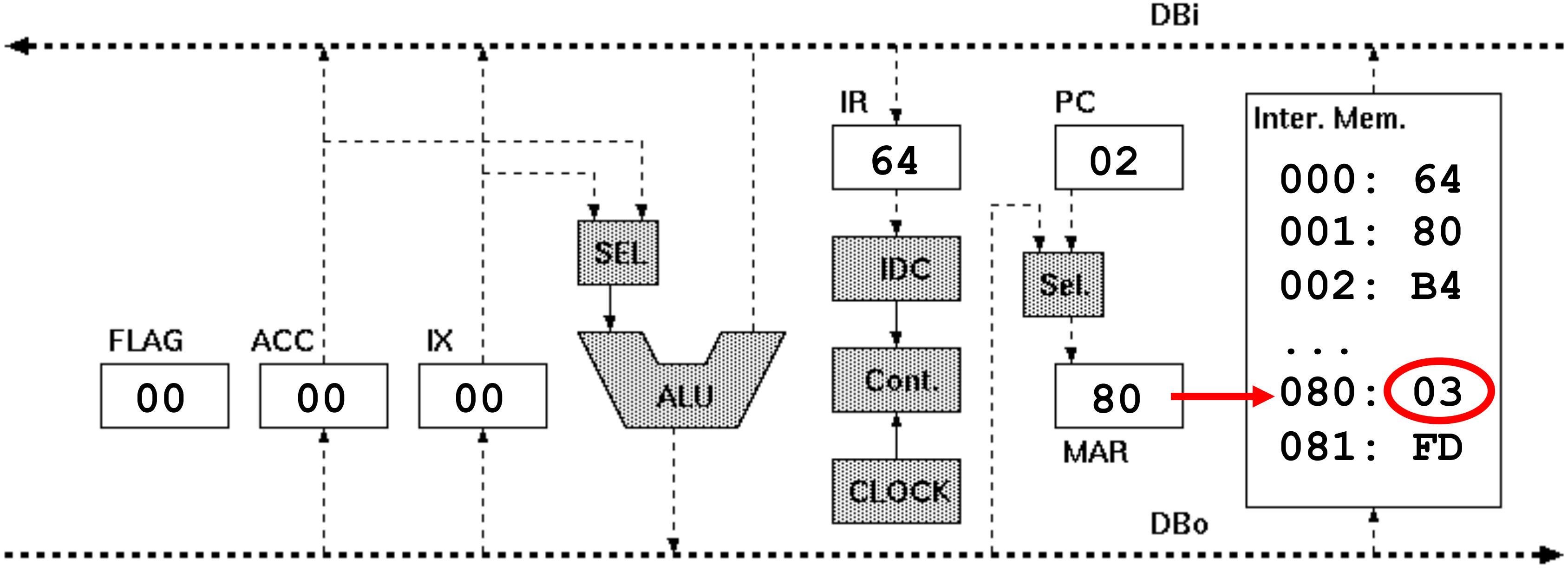
Trace of the execution

LD ACC,[D1] P3: (Mem)→MAR



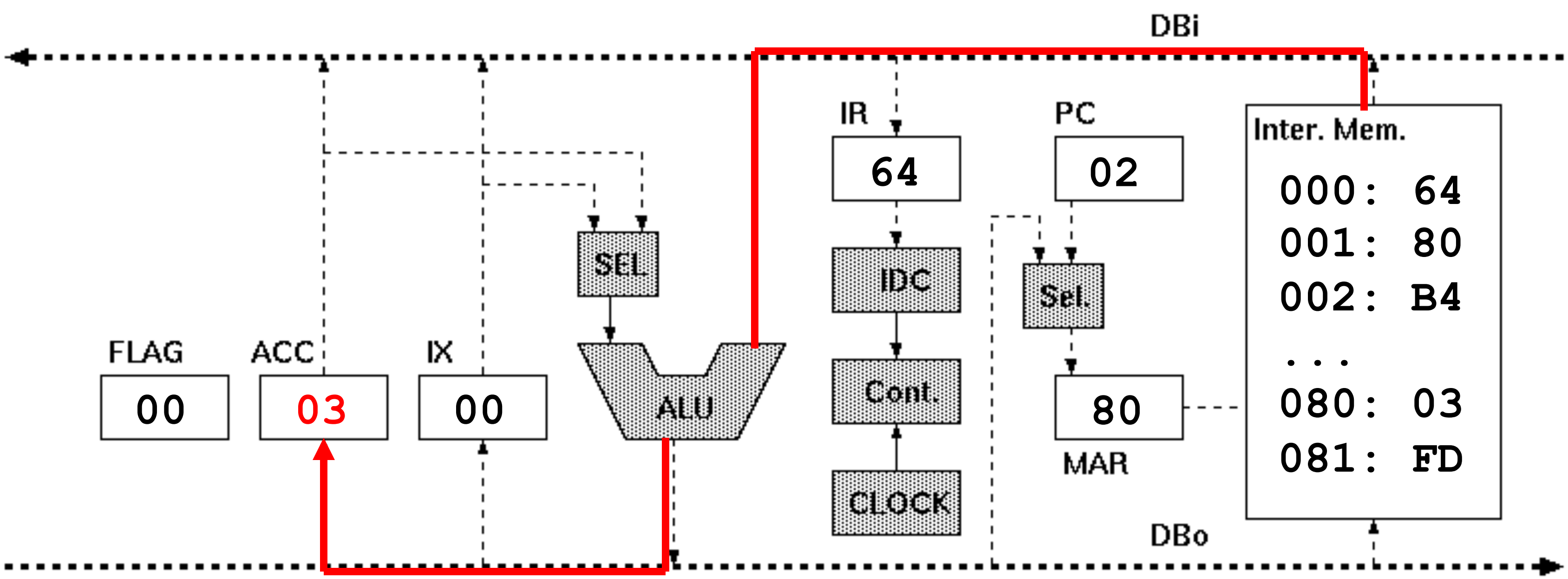
Trace of the execution

LD ACC,[D1] P4: (Mem)→A



Trace of the execution

LD ACC,[D1]　　　P4: (Mem)→A



Flag register

- Carry Flag, CF (桁上がりフラグ)
  - If carry-over occurs, CF = 1.
- Overflow Flag, VF (桁あふれフラグ)
  - If over-flow happens, VF = 1.
- Negative Flag, NF (負フラグ)
  - If the result if negative, NF = 1
- Zero Flag, ZF (ゼロフラグ)
  - If the result is zero, ZF = 1.

p.22 Fig. 2

Problem 3.1 (p.33)

- (1)
  - Trace the observable registers and buses during the beginning and end of the execution.
- (2)--(6)
  - Trace the flag register during the beginning and end of the `ADD` command.
  - Change `ADD` to `ADC` and trace the flag register during the beginning and end of the `ADD` command.
  - Record the results of each addition.

Problem 3.1: Caution 1/2

- “64” in hexadecimal, ???????? in binary
- To input a value in the 80th address, the MAR should first be operated upon.
- Check the result at the beginning
- Pay attention to avoid misreading “6” as “b.”

Problem 3.1: Caution 2/2

- Use “two’s complement (2の補数表現)” for negative values

	3		0	0	0	0	0	0	1	1
+	−3		1	1	1	1	1	1	0	1
		1	0	0	0	0	0	0	0	0

Points for report

- (1)
  - Explain the operation in each phase of all commands with sentences and figures.
    - You can refer pp. 24--28.
    - You can download some material here.
    - https://expcs.github.io/microprocessor/
- (2)--(6)
  - What is the condition for changing each of the flags.
  - Explain the differences between `ADD` and `ADC`.

What we will do today

- Introduction
- Fundamental usage of KUE-CHIP2
- Problem 3.1
  - Trace the values in `ACC`, `PC`, `FLAG`, etc., while executing `ADD` and `ADC`.
- Problem 3.3 (1)
  - Check the clock frequency
  - Generate a single tone that are as much accurate 440 Hz as possible.
- Introduction of next problem

Output a melody

- Output waves from KUE-CHIP2 to generate a sound from a speaker.

- Today: the basic mechanisms to generate a sound
- 3rd lecture: run a program to output a melody

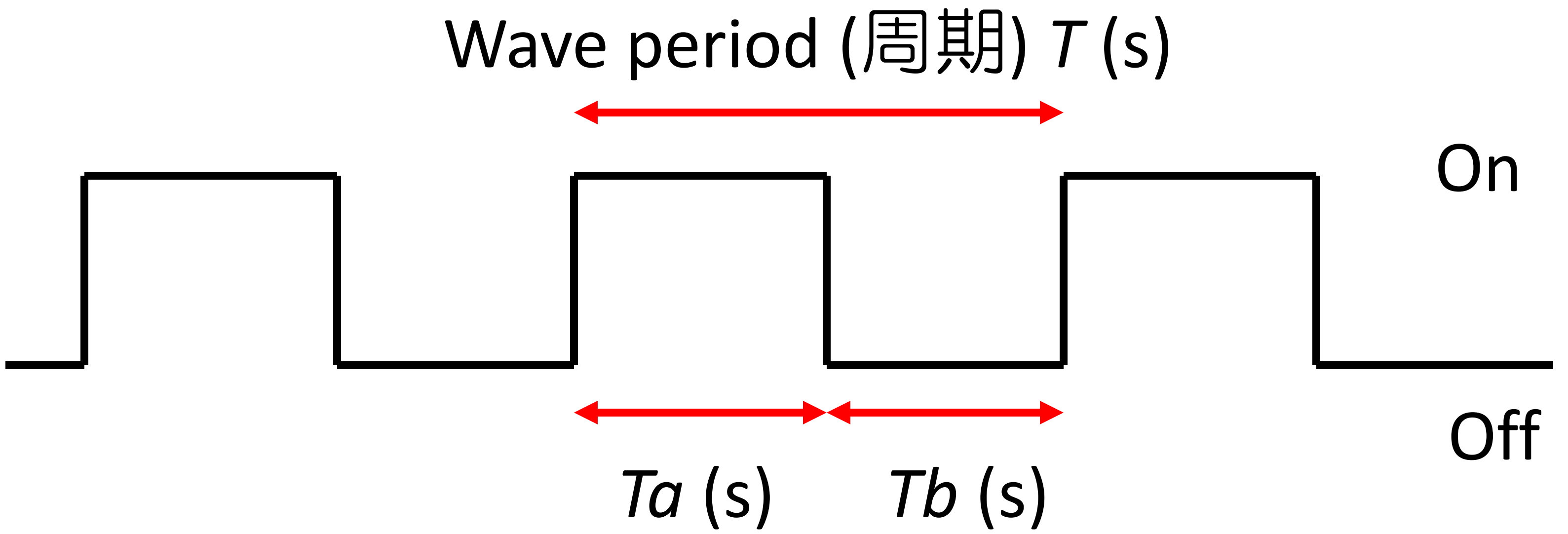
What is sound?

- Sound is vibration (waves) that travels through the air.
- There are three elements of sound:
  - Loudness: The amplitude of the wave
  - Pitch: The frequency of the wave
  - Timbre: The harmonic content of a sound

- Loudspeaker:
  An electroacoustic device that converts electric signals to vibrations of air (sound).

Waves to generate

- Rectangular wave



- T ＝ Ta ＋ Tb

Wave generation (p.39, List 4)

Address	label	instruction	operand	# of phases
00:	L0:	LD	ACC, FFh	4
02:		OUT		4
03:		LD	ACC, *a*	4
05:	L1:	SUB	ACC, 01h	4
07:		BNZ	L1	4
09:		LD	ACC, 00h	4
0B:		OUT		4
0C:		LD	ACC, *b*	4
0E:	L2:	SUB	ACC, 01h	4
10:		BNZ	L2	4
12:		BA	L0	4

Determine by yourself

Determine by yourself

Wave generation (p.39, List 4)

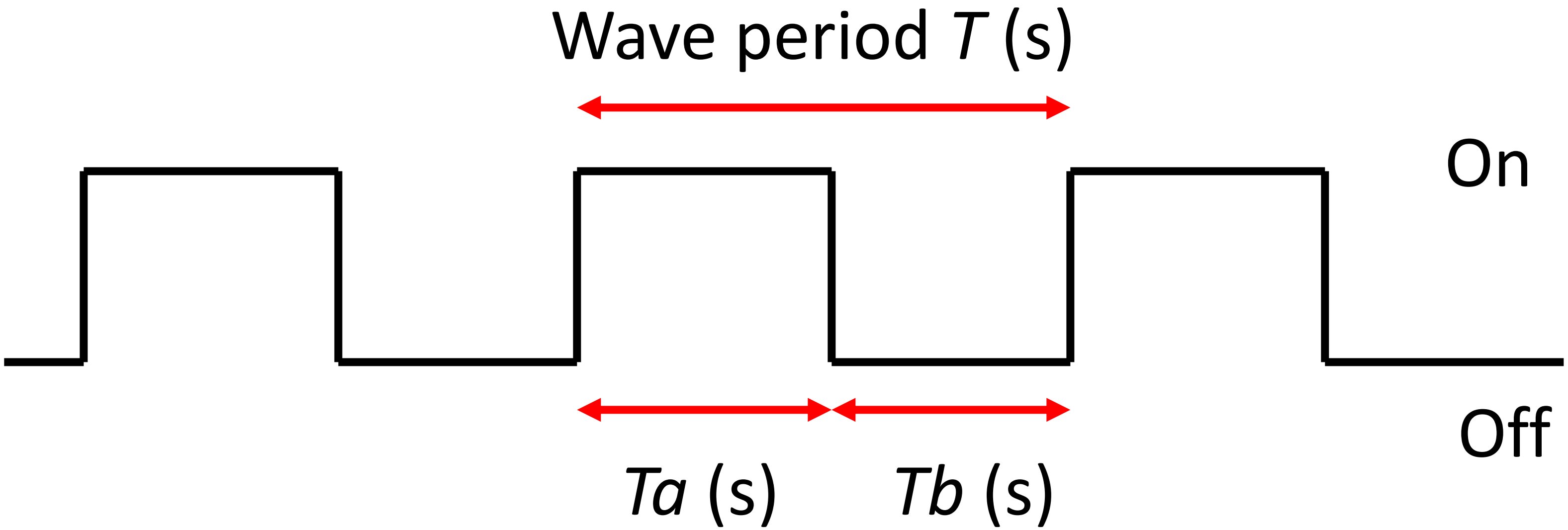
Address	label	instruction	operand	# of phases
00:	**L0:**	**LD**	**ACC, FFh**	**4**
02:		**OUT**		**4**
03:		**LD**	**ACC, *a***	**4**
05:	**L1:**	**SUB**	**ACC, 01h**	**4**
07:		**BNZ**	**L1**	**4**
09:		**LD**	**ACC, 00h**	**4**
0B:		**OUT**		**4**
0C:		**LD**	**ACC, *b***	**4**
0E:	**L2:**	**SUB**	**ACC, 01h**	**4**
10:		**BNZ**	**L2**	**4**
12:		**BA**	**L0**	**4**

"On" part of the wave (addresses 00–07)

"Off" part of the wave (addresses 09–10)

Waves to generate

- Rectangular wave



- T ＝ Ta ＋ Tb
- In the list 4, Ta ＝ (12+8a)T0, Tb ＝ (16+8b)T0
  (where T0 = time for 1 clock)

Problem 3.3 (1) p.33

- (a) Examine the period for one clock
  - Set the `CLK` switch to the middle
  - Set the `CLKFRQ` dial to 0 to 8. Measure the frequency for each.
  - The signal is output from `JP3` (at the second highest row of the right column)
- (b) Determine a and b in the list 4.
  - The frequency to output: 440Hz "A"
  - Determine the optimal *T0, a,* and *b* by calculation
    - T = Ta + Tb, T = 1/440 (s)
    - Ta = (12+8a)T0, Tb = (16+8b)T0

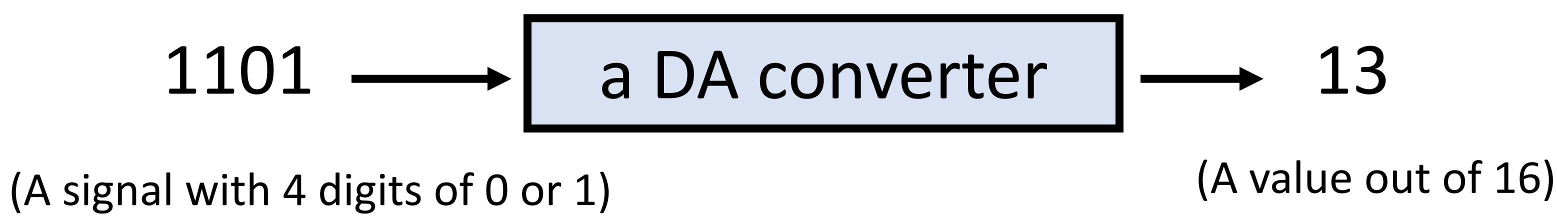
Problem 3.3 (1) p.33

- (c) Output a wave of frequency 440 Hz
  - Input the program from list 4
  - Set the `CLKFRQ` dial
  - Measure the frequency of the signal on the oscilloscope through DAC.'
  - Confirm, through calculation, that the output frequency is 440 Hz with an error of ±1%.

Digital to analogue value

- Attach a DA converter to the output buffers, and send output signals to the oscilloscope.

- A DA converter (DAC):
- It is an electronic device that takes a digital numerical value as input, and outputs a voltage signal based on the input.



Notes for the DAC

- The DAC is fragile.
- Treat it carefully. (Don’t touch it needlessly.)
- Take special care of the circuit around the attachment part.
- The lecturer / TA will attach or detach it for you.

How to connect DAC

- Connect the DAC to the oscilloscope;
  channel 1 → Red
  channel 2 → Blue
  ground → Black
- Set `CLKFRQ` dial to "1" and run the program.

Notes for your report for (3) Output a melody (a) Ensure that the accuracy is within a range of ±1%.

- How did you determine the optimal *T0*, *a*, *b*?
  - Describe the calculation process.
- How did you check it?
  - Calculate the error between the generated and target frequency.
- Are there any other ways to check the error?

Notes for your report for (3) Output a melody (b) Propose methods for increasing the accuracy

- Involving the KUE-CHIP2 only (by the software schemes)
- Connecting KUE-CHIP2 with some device (by hardware schemes)

※ Consider methods of making the output frequency close to 440 Hz while still using the same algorithm to generate the sound.

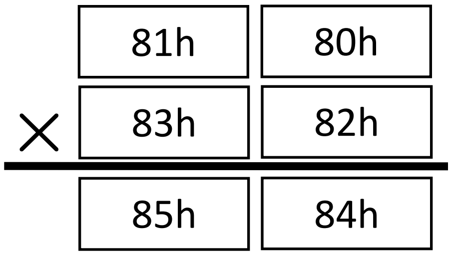
What we will do today

- Introduction
- Fundamental usage of KUE-CHIP2
- Problem 3.1
  - Trace the values in `ACC`, `PC`, `FLAG`, etc., while executing `ADD` and `ADC`.
- Problem 3.3 (1)
  - Check the clock frequency
  - Generate a single tone that are as much accurate 440 Hz as possible.
- Introduction of next problem

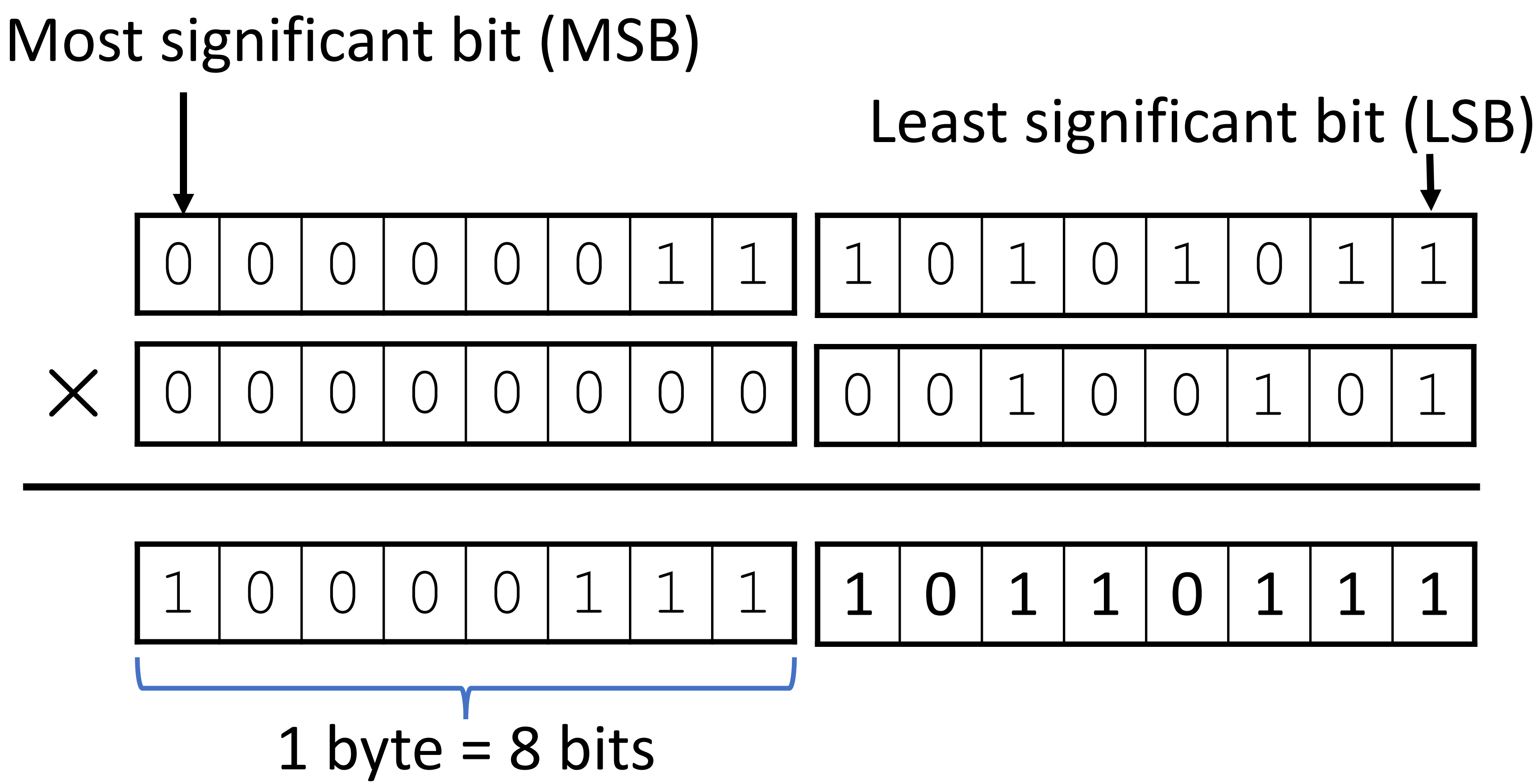
Next class: Problem 3.2: Multiplication

- Multiplication of 2 bytes precision level values without signs
- You do not have to store the data in the addresses shown in the text.
- You can assume the result is within 2 bytes

- Preparations
  Prepare and assemble a program



Supplementary:
2 bytes-precision level multiplication



Be cautious of the addresses on the memory

Supplementary: Address modes

- They are ways for notations of operands
- The address modes for KUE-CHIP2 (pp.29–31) are:
  - `ACC`, `IX`: the content in `ACC` (`IX`) is data
  - Immediate: the operand itself is data
  - Direct: the operand is the address and the content of the address is data
  - Indirect: “the operand + the content of `IX`” is the address and the content of the address is data.

Supplementary: ADD, ADC, RCF (p.24)

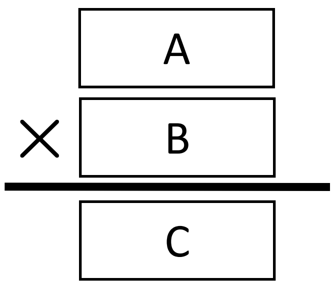
- ADD: It adds the two operands without CF
- ADC: It adds the two operands with CF
- SUB and SBC also have the same relationship as ADD and ADC.

- RCF: It resets the CF

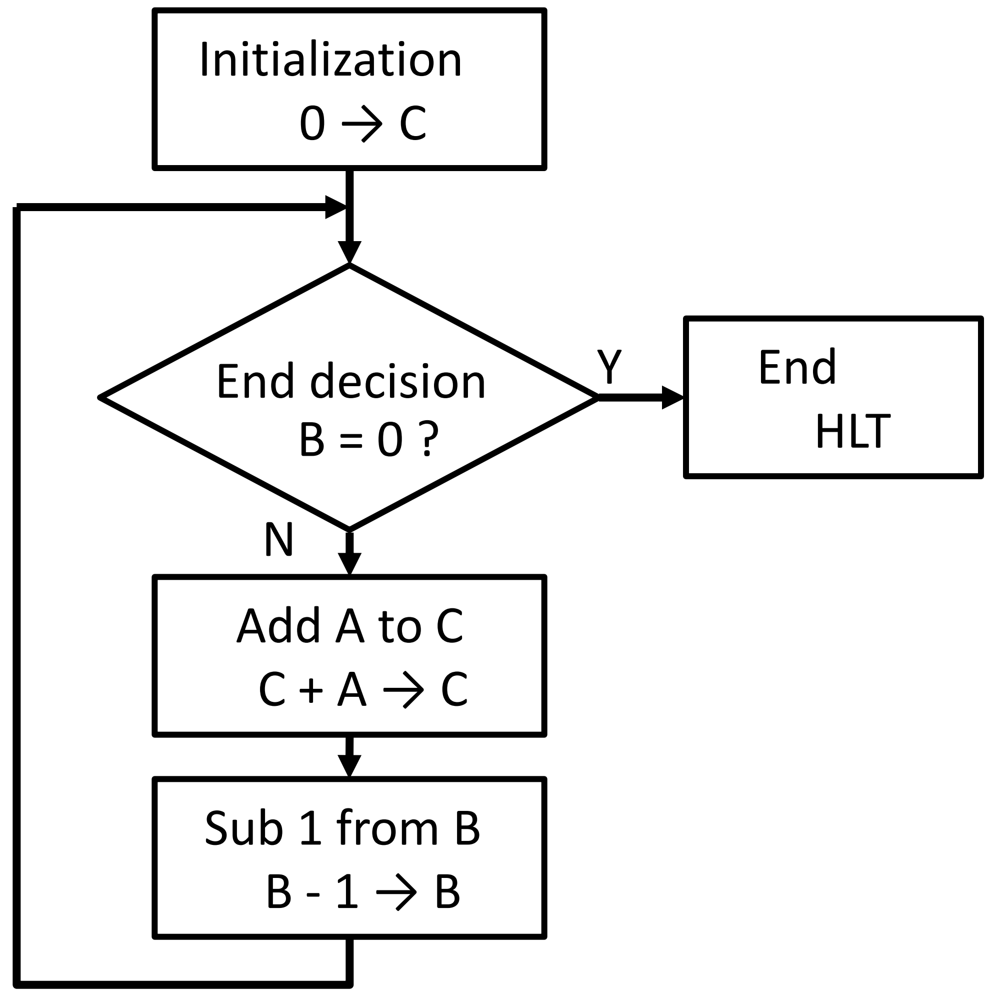
Notes

- You should prepare a program to begin immediately the program input.
- You should make a flowchart before writing codes.
  - You can easily find bugs with the help of the flowchart.
  - It is recommended that the programs and the flowcharts are printed on separate pages.
- We will not help if you are late or do not prepare them.
  - Your points will be taken off or we will not accept your report.

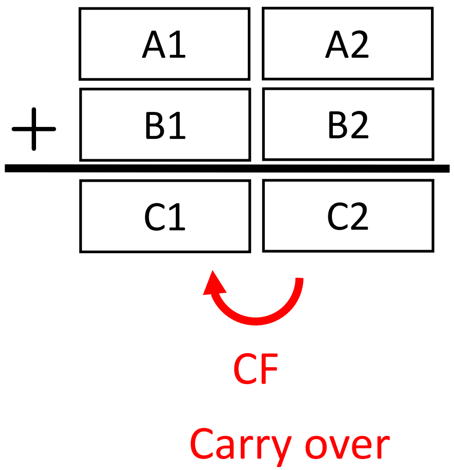
Example of flowchart
Multiplication with 1 byte precision level

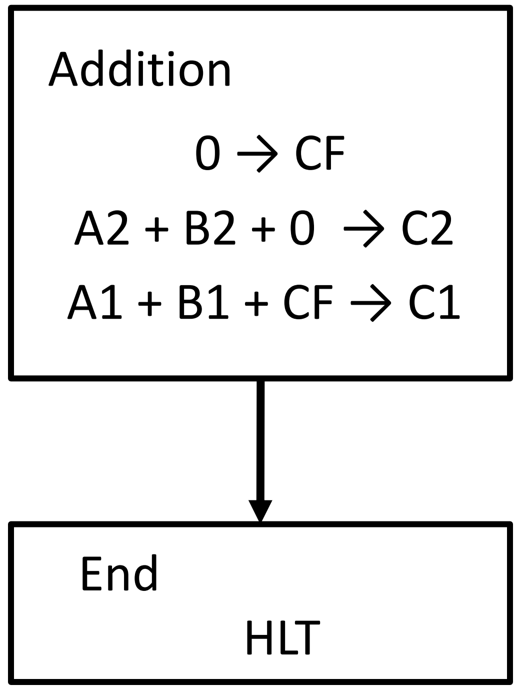


The flowcharts illustrate the flow of programs by figures and sentences



Example of addition with 2 byte precision level






```
RCF

LD      ACC,  [A2]
ADC     ACC,  [B2]
ST      ACC,  [C2]

LD      ACC,  [A1]
ADC     ACC,  [B1]
ST      ACC,  [C1]

HLT
```

Notes for making programs

- Make the program understandable to others
- Print them out
- Do not use both sides of the papers
- Arrange assembly and machine languages neatly
- Leave spaces for modification
- Use of both of binary and hexadecimal is fine, but hexadecimal is useful to check the program on the board

Arrange assembly and machine languages neatly

```
000:  20          RCF

001:  64  80      LD    ACC, [A2]
003:  94  82      ADC   ACC, [B2]
005:  74  84      ST    ACC, [C2]

007:  64  81      LD    ACC, [A1]
009:  94  83      ADC   ACC, [B1]
00B:  74  85      ST    ACC, [C1]

00D:  08          HLT
```

address

Frequently occurring errors

- The addresses for 2-byte data
  - Check which addresses if both high and low bytes are used
- Initialization
  - `SUM += A`
- Carry over (ADD, ADC, RCF)
- Decision for the end
  - Do not flag `ZeroFlag` if “`LD 0`”
- Forget to store the results
- Addresses are represented in decimal notation

Emulators for the preparation

- A KUE-CHIP2 Emulator http://www.vector.co.jp/soft/winnt/util/se506103.html
- A KUE-CHIP2 web assembler http://www.hpc.se.ritsumei.ac.jp/kue-chip2/kue2-webasm/
- KEMU Emulator (←Recommended)
- https://emu.kemuide.openwaseda.net

What we will do today

- Introduction
- Fundamental usage of KUE-CHIP2
- Problem 3.1
  - Trace the values in `ACC`, `PC`, `FLAG`, etc., while executing `ADD` and `ADC`.
- Problem 3.3 (1)
  - Check the clock frequency
  - Generate a single tone that are as much accurate 440 Hz as possible.
- Introduction of next problem

Microprocessors (Lecture 2)

Lecture 2

- Problem 3.2: Creation of a multiplication program
- Write a program to multiply two unsigned 2-byte numbers
- The addresses for storing data are not necessarily the same as in the textbook
- You can assume the results will stay within a length of 2 bytes

- Preparation required:
  Writing a program and assembling it

Frequent mistakes (reshown)

	81h	80h
×	83h	82h
	85h	84h

- Handling 2-byte data
  - Mistook upper or lower addresses
- Missed initialization
  - `SUM += A`
- Failure of carry (`ADD`, `ADC`, `RCF`)
- Wrong judgment of completion
  - `LD 0` does not reset the Zero Flag.
- Forgot data storing (`ST`)
- Addresses are in decimal (not in hexadecimal)
- Typing mistakes, wrong assemblies

Procedure

- Input your program into the board.
  - We will check the flow chart during input.
- Perform operation checks for examples 1 to 4 written on the white board.
- Then calculate A and B on the white board. Measure the execution time (at 100 Hz).
- Fill the execution time and memory usage (unit: bytes) on the white board.
  - Memory usage = memory for program + for storage

Theoretical execution time

- Calculate the theoretical execution time for your program, and compare it with the actual time as follows;
  1. Identify the parameters needed to determine the theoretical execution time.
     - The number of phases for each instruction (see Table 2 on p.18)
     - 1 phase = 1 clock
     - Clock frequency = 100 Hz
  2. Derive an expression for the calculation.
  3. Calculate the time by using the expression.
  4. Compare the time.

Notes for your report

- Explain your program with a flow-chart.
- Compare your program to that of at least two other students. Points to include in the comparison are:
  1. actual execution time
  2. memory consumption

- Note that you do not have to insert the others' program lists, but describe them briefly.

For the next lecture
Problem 3.4 (2) Output a melody

- Preparation required:
  writing a program and assembling it
- Reference: Appendix B.2 and list 5 (p.41)
  - The preparation of only the data for music score does not complete it.
  - Some modifications are needed in list 5.
- If you cannot complete the problem in time, it will just be closed.
  We can support you only if you have prepared a program.

Notes

- Output the melody as an endless loop
- Don't make any sound outside the audible range
- Refer Table 13 on p.40
  - One octave higher → Double the frequency
- Some modifications are needed in list 5.
  - How can you represent a "rest"?
    → Distinguish between a note and a rest to process them differently.
  - When the same notes continue, they are heard as one long note.
    → A space is needed between the notes in this case.

Generation of a melody (list 5)

Program region

```
000:  62 00         LD ACC, dptr1
002:  75 1A         ST ACC, (dptr)

004:  65 1A    L0:  LD ACC, (dptr)
006:  68            LD IX, ACC
007:  B2 03         ADD ACC, 0x3
009:  75 1A         ST  ACC, (dptr)

00B:  A2 18         SUB ACC, dptr2
00D:  31 13         BNZ L1
00F:  62 00         LD ACC, dptr1
011:  75 1A         ST ACC, (dptr)

013:  67 02    L1:  LD ACC, (IX+2)
015:  75 1C         ST ACC, (n3)
```

The addresses 00, 1A, 1C, and 18 in the list represent the ones you are using. (Change it to fit your program)

Data region

```
100:  n1 n2 n3   dptr1:  C
103:  n1 n2 n3           D
106:  n1 n2 n3           E
109:  n1 n2 n3           F
10C:  n1 n2 n3           G
10F:  n1 n2 n3           A
112:  n1 n2 n3           B
115:  n1 n2 n3           C
118:             dptr2:  (not used)
119:  00 or ff           image
11A:  ??                 dptr
11B:  ??                 n2
11C:  ??                 n3
```

The start (dptr1)

The end (dptr2)

Output (image)

Used when running (11A–11C)

It points to the current sound (increase by 3)

n1 changes the pitch, and n2 and n3 change the length of the tone (double loop)

Microprocessors (Lecture 3)

Problem 3.3 (2) Output a melody

- Output a simple melody
- Preparation required:
  writing a program and assembling it
- Reference: Appendix B.2 and list 5 (p.41)

- Take care when you handle a DAC.

Notes for your report
(4)(c) Describe the method of data expression

- Use a program list

- For example, a musical score represents a melody in a readable expression for humans.
- What is the understandable expression of data for this case?

Notes for your report
(4)(d) Can you use the same way of outputting the melody for other CPUs?

- Give an example of a CPU.
- Study the instructions of the CPU you choose.

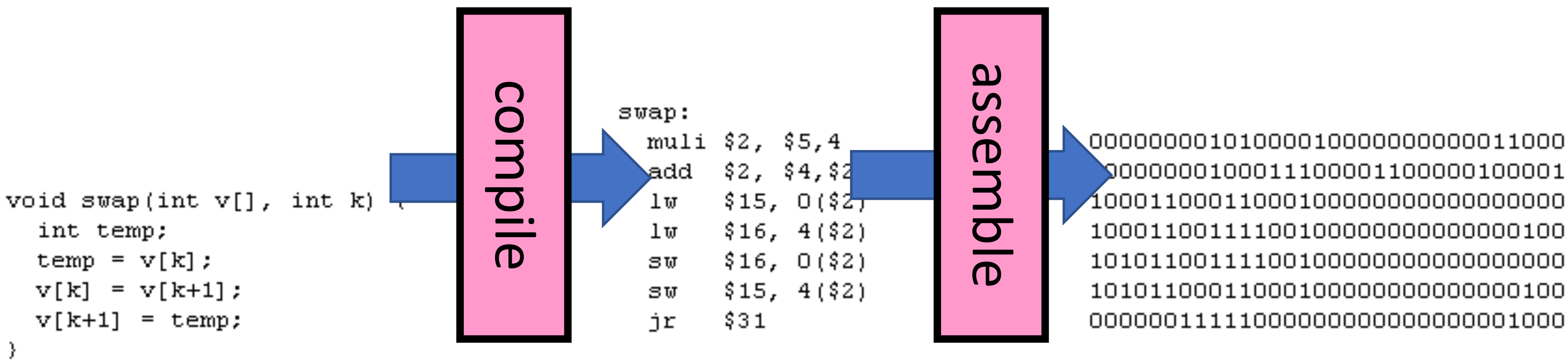
Notes for your report
(5) Study the CPU you mostly use (or a famous CPU) and its architecture, and report it.

- Include the features of registers, instructions, or memory spaces, etc.
- How to execute a multiplicative instruction on the CPU?
- Arbitrary problem
  - You do not need to do, but the problem will be additionally scored.

Summary

- We have learned about the mechanism through which a computer works.
  - For example: why can't a 32 bit OS handle more than 4 GB of memory?
- We learned how to program and debug.

$4G = 4 \times 1024 \times 1024 \times 1024$
$= 2^2 \times 2^{10} \times 2^{10} \times 2^{10} = 2^{32}$



High-level language | Assembly language | Machine language

Report submission 1/3

- Read the requirement written on page 6 of the textbook carefully.
- Send your report as a PDF file to fukumura@cs.tut.ac.jp.
- You can make your own format for the cover.
- No need to copy the description of the experimental methods straight from the textbook.
- Refer "Notes for your report" on this PPT file.
- You must check your report by using the self inspection sheet. (No need to submit the sheet.)

Report submission 2/3

- The deadline is on 23.59 one week after today (be punctual)
  - No extension of the deadline is acceptable except in case of an accident or illness.
  - You can submit an improvement or modification of the report until one week after the deadline.
  - We will not accept uncompleted reports (half done).

Report submission 3/3

- The subject of the E-mail:
  - [report] [student ID] [your name]
  - [レポート] B123456 豊橋太郎
- Convert the report to a PDF file.
- The name of the pdf file should be:
  - [Your school register number]-[your name].pdf.
  - B123456-豊橋太郎.pdf
- We will respond to you within 3 days. If you do not receive a reply from us after 4 days, come to room F-408.
- If you have any question, please ask me.