Microprocessor（Lecture 1）

Introduction

- 情報・知能工学系 学生実験サイト
  http://www.cs.tut.ac.jp/jikken/
- 実験の説明資料などはWeb上で公開中
  https://expcs.github.io/microprocessor/
- 実験レポートの受け取りはメールにて行う
  fukumura@cs.tut.ac.jp
  - レポートは日本語でも英語でも可
- 質問がある場合はF-408を訪問するか，E-mailで回答します.

日程 (see p. 26)

Week 1

Lecture 1: イントロダクション
Problem 3.1: 加算
Problem 3.3 (1): 単音の出力

Week 2

Lecture 2: Basic Programming
Problem 3.2: 乗算

Week 3

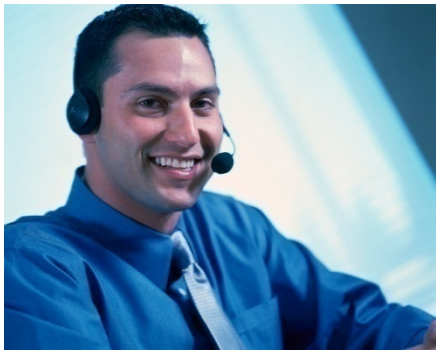
Lecture 4: Applied programming
Problem 3.3 (2): メロディの出力

第2回以降は予習(プログラムの準備)が必須

今日やること

- 導入
- KUE-CHIP2の基本的な使い方
- Problem 3.1
  - ADDとADCを実行しながら，ACC, PC, FLAGなどの値を記録する.
- Problem 3.3 (1)
  - クロック周波数を記録する
  - できるだけ440 Hzに近い単音を出力する
- 次の課題の説明

Relationships between a computer and a user

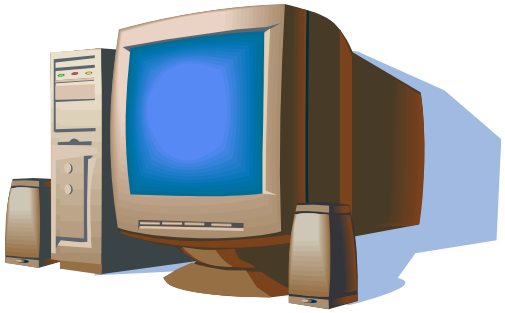


User

Input

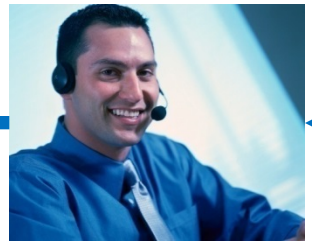
どのような仕組みで
動いているのか？

Output



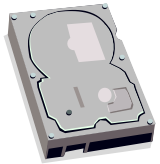
Computer

Hardware



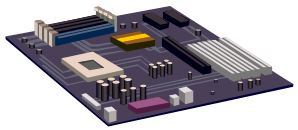
Input devices

Output deivices

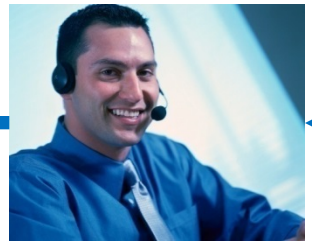


Storage

Processing Unit



Software

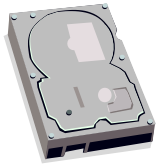


Input devices

Output deivices

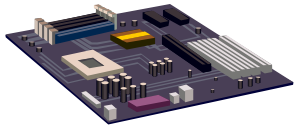
Application program

System program



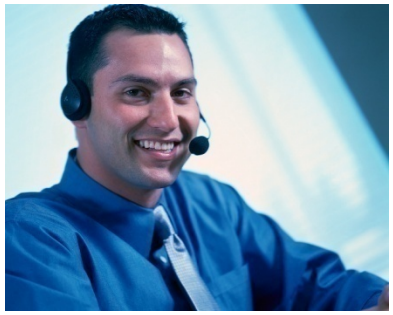
Storage

Processing Unit



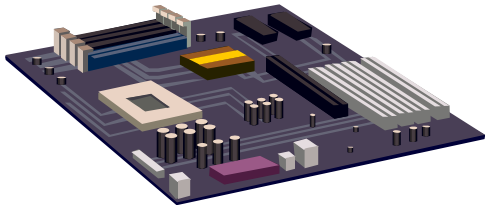
Question

- 処理装置 (CPU) はプログラム(≒ソフトウェア)をどのように解釈しているのか?



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

高級言語による
プログラム

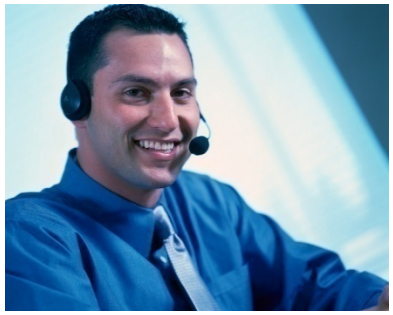


```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

機械語による
プログラム

(今のところの) 回答

- コンパイラ，アセンブラと呼ばれるプログラムを変換するプログラムを使う

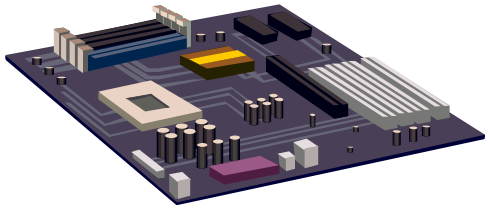


```
void swap(int v[], int k)
{
  int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

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
```

assemble



```
00000000101000010000000000011000
00000001000111000011000001000001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

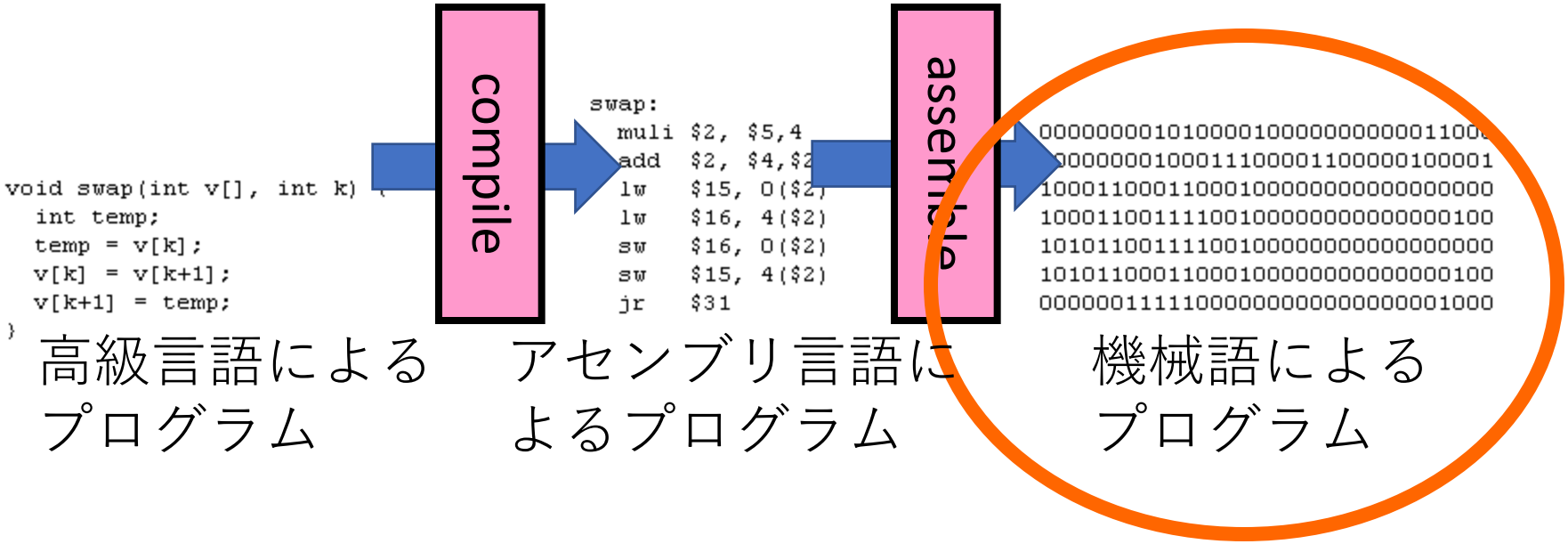
高級言語によるプログラム

アセンブリ言語によるプログラム

機械語によるプログラム

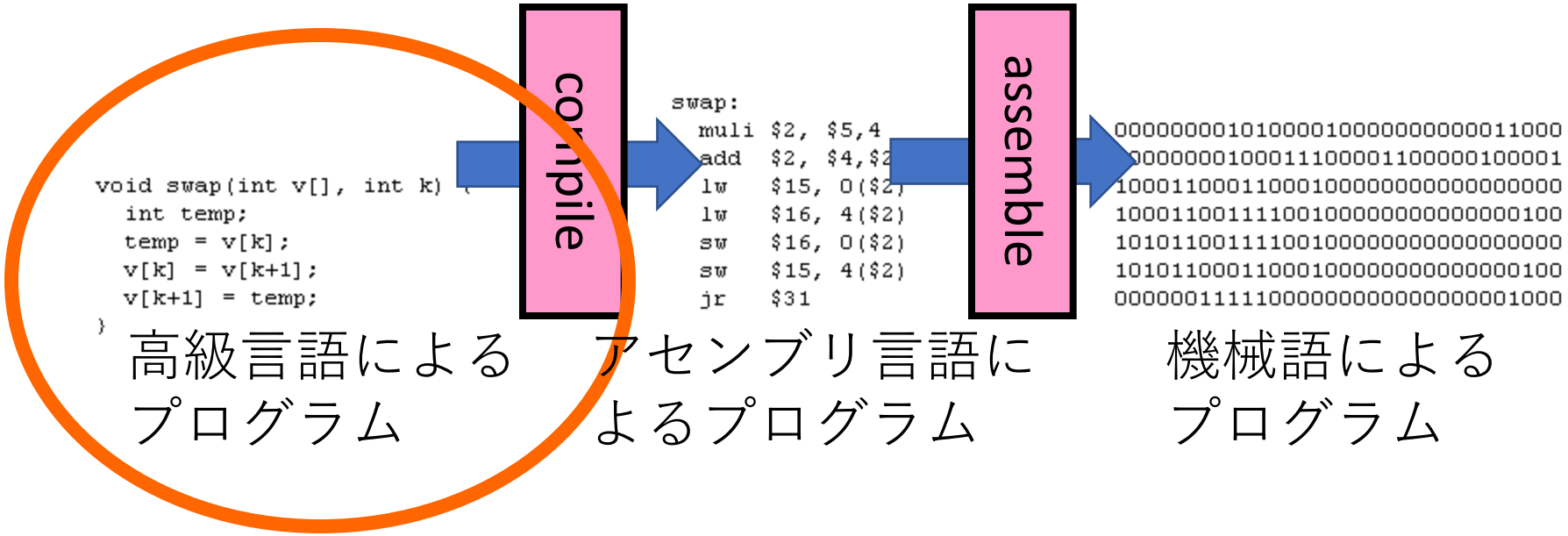
機械語とは何か？

- CPUが直接理解し実行できる言語のこと
- プログラムは0と1の列から構成される
- CPUごとに異なる



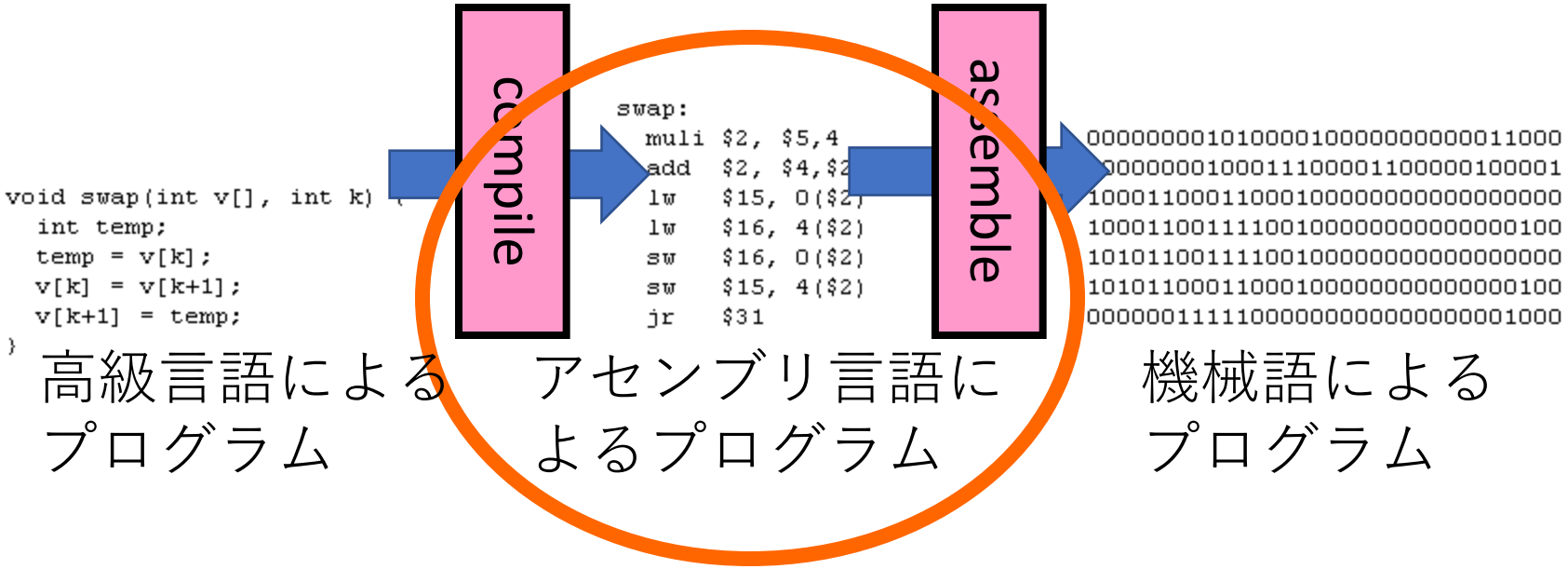
高級言語とは何か？

- 人間が使いやすいプログラミング言語
- C，C++，Java，Perlなど
- CPUごとに異ならない（共通）



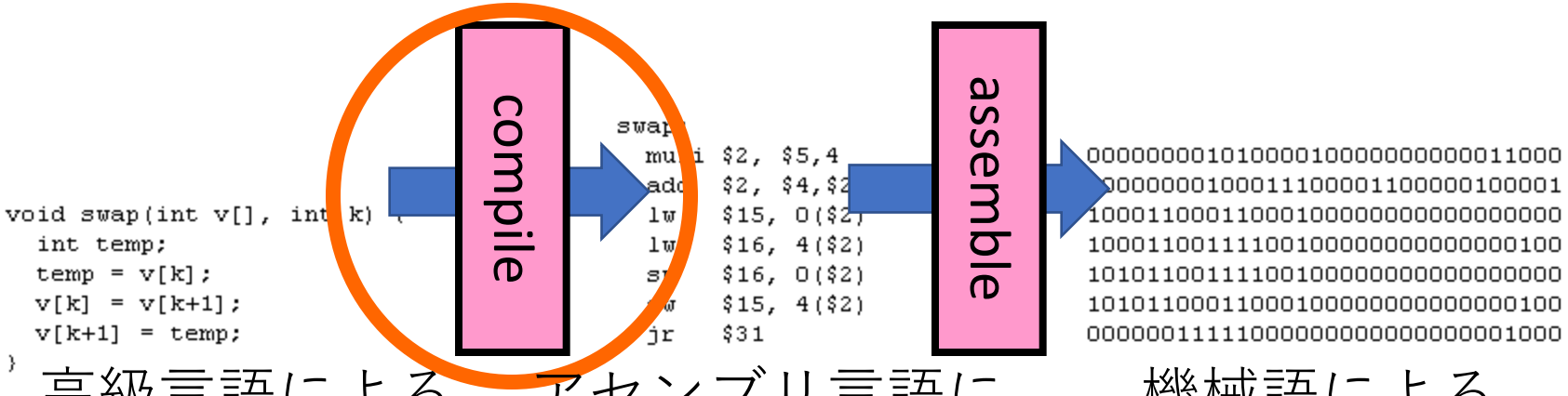
アセンブリ言語とは何か？

- 機械語を人間にわかりやすくした言語
- 命令が機械語と (ほぼ) 1 対 1 に対応
- CPUごとに異なる



コンパイルとは何か？

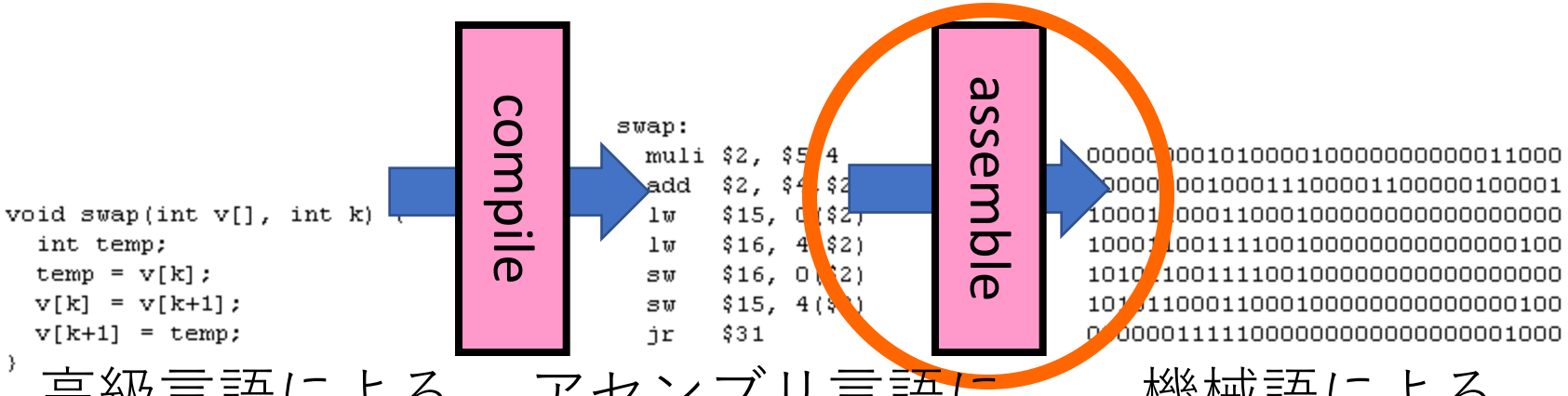
- 高級言語によるプログラムをアセンブリ言語によるプログラムに（または機械語によるプログラムに）翻訳すること



高級言語によるプログラム　　アセンブリ言語によるプログラム　　機械語によるプログラム

アセンブルとは何か？

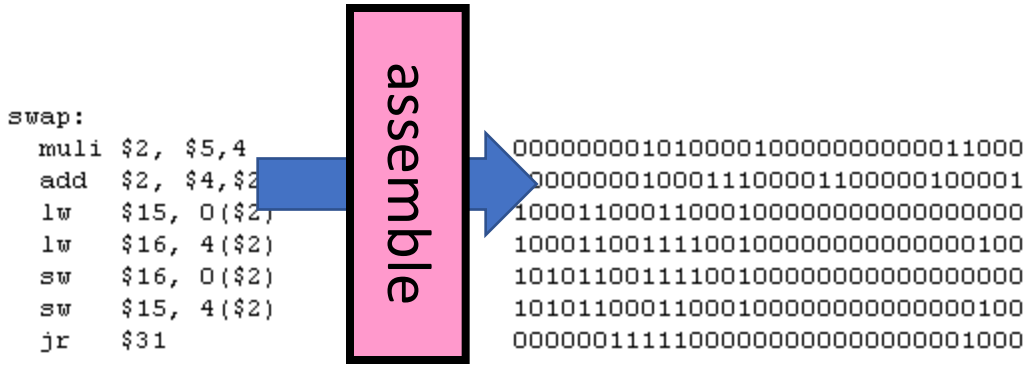
- アセンブリ言語によるプログラムを機械語によるプログラムに翻訳すること



高級言語によるプログラム　　アセンブリ言語によるプログラム　　機械語によるプログラム

この実験の流れ

1. アセンブリ言語によるプログラミング
2. 作成したプログラムのアセンブル(手作業)
3. 実行(動作の理解)



アセンブリ言語によるプログラム　　機械語によるプログラム

Device used in this theme

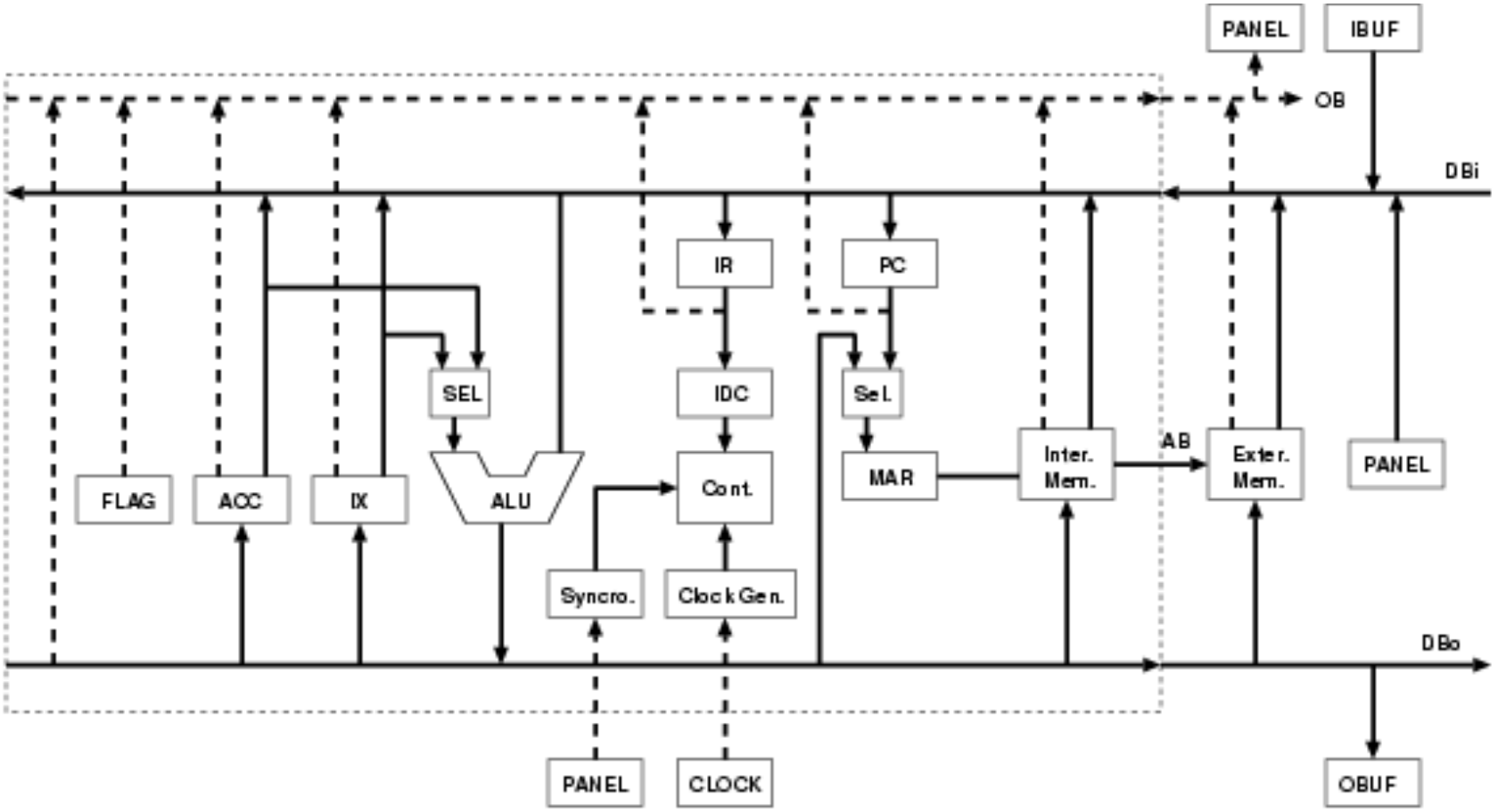
- KUE-CHIP2
- 教育用の8ビットマイクロプロセッサ
  = CPU

8 bits = 1 byte

0	0	0	1	0	0	1	1

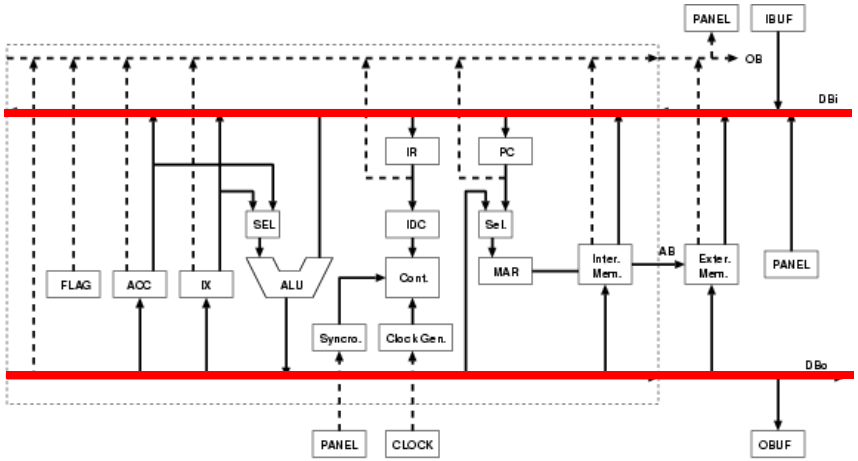
13h ← 16進数であることを示す
他にも13H，0x13など

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



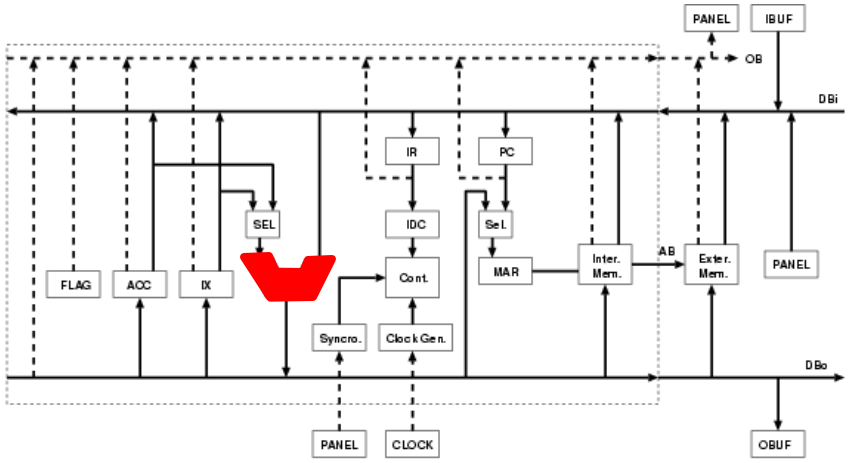
KUE-CHIP2: bus

- 入力バス：入力部分とCPU内部を結ぶ
- 出力バス：出力部分とCPU内部を結ぶ



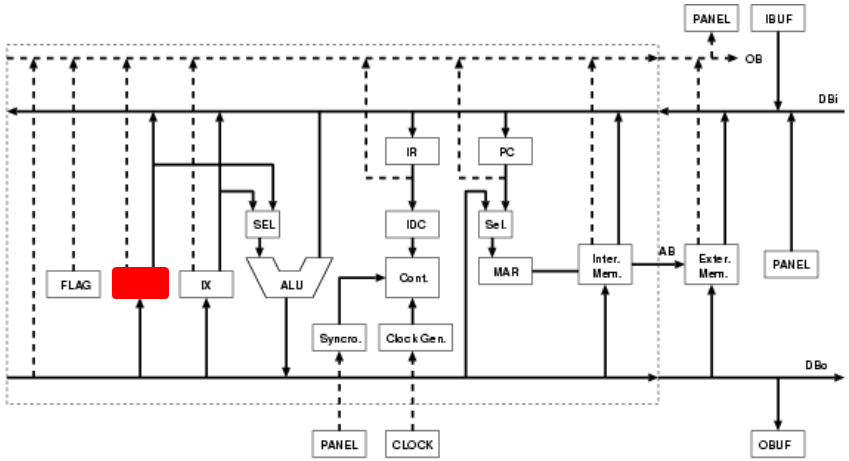
KUE-CHIP2: ALU

- 演算ユニット(Arithmetic and Logic Unit)
- 算術演算，論理演算，アドレスの計算を行う



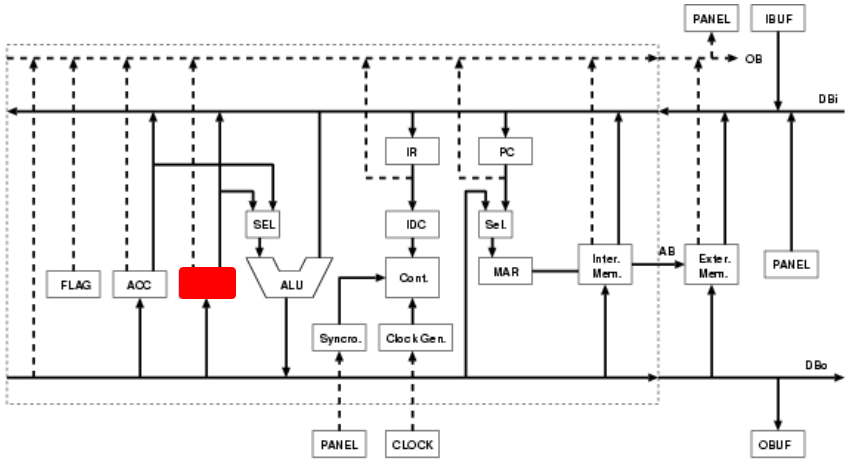
KUE-CHIP2: ACC

- アキュムレータ (accumulator)
- 演算に利用するレジスタ．8ビット
- 演算対象，演算結果を保持



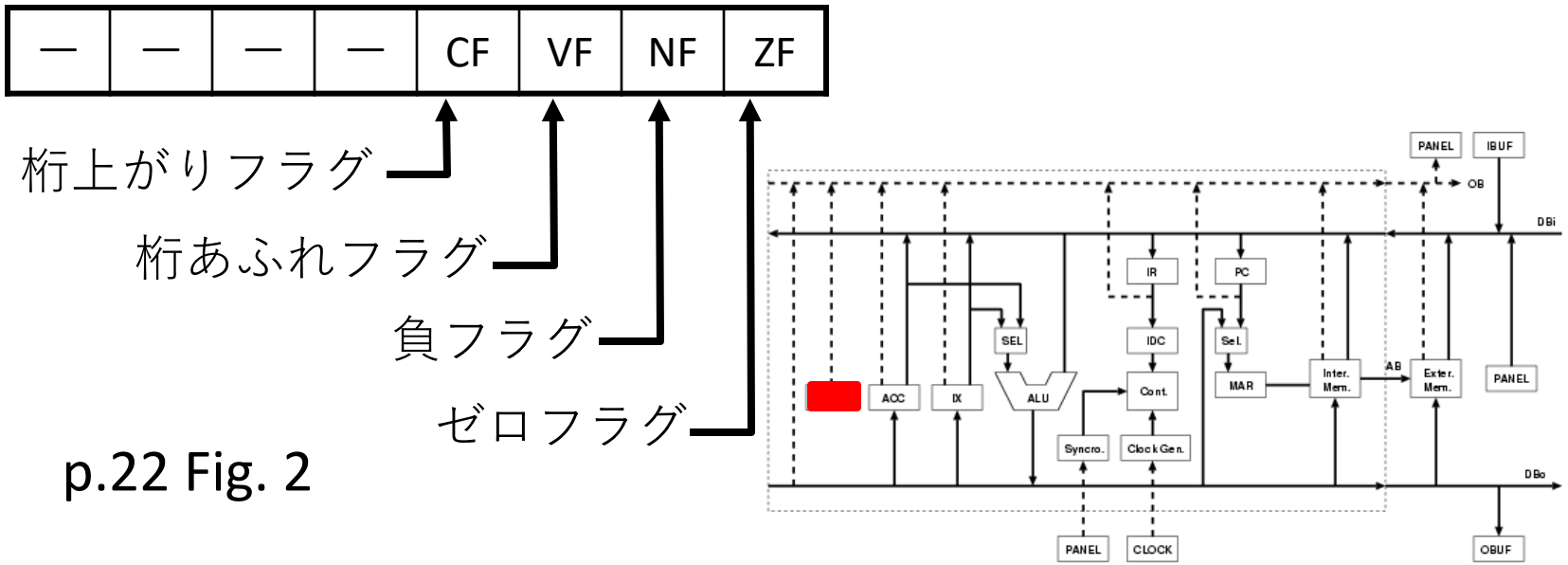
KUE-CHIP2: IX

- インデックスレジスタ (index register)
- 演算に利用するレジスタ．8ビット
- 演算対象，演算結果を保持
- 修飾アドレス指定のときのアドレス修飾にも使用



KUE-CHIP2: FLAG

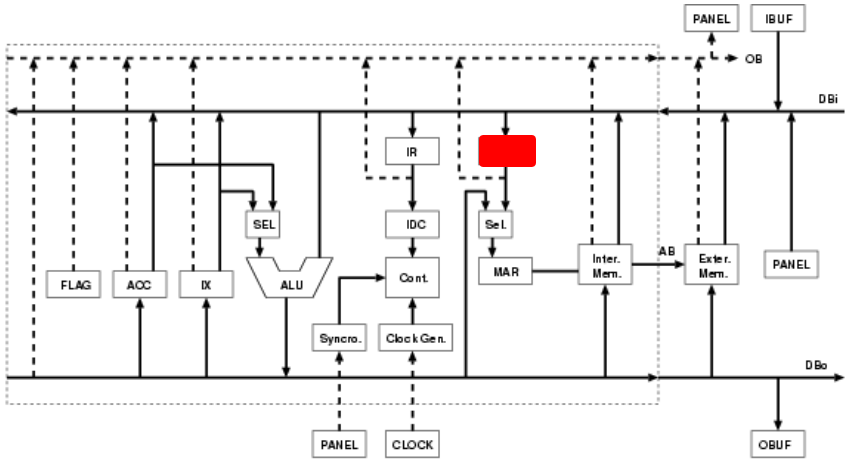
- Flag register
- 演算・シフト結果により変化．4ビット



p.22 Fig. 2

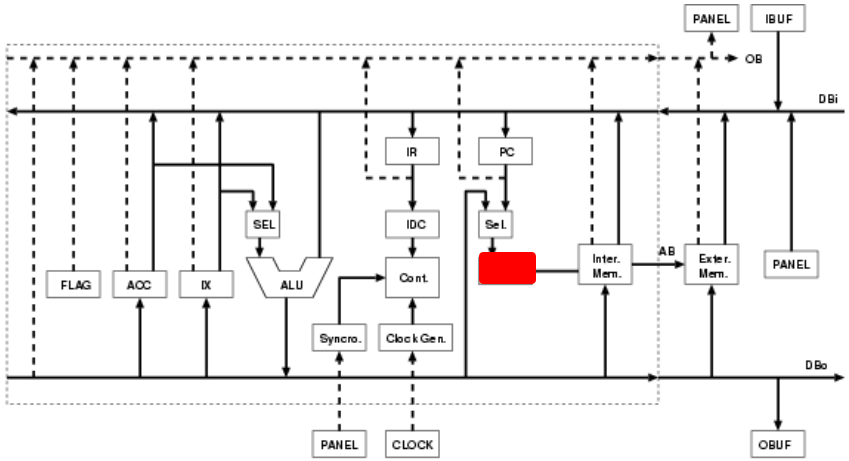
KUE-CHIP2: PC

- プログラムカウンタ (program counter)
- 次に実行する命令のメモリ上での
- アドレスを保持．8ビット



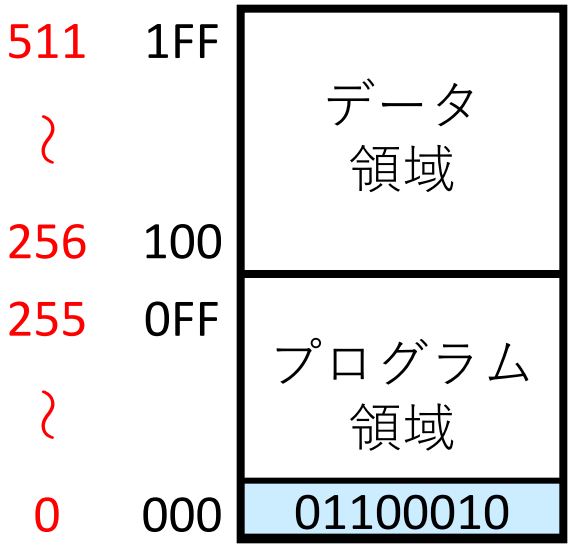
KUE-CHIP2: MAR

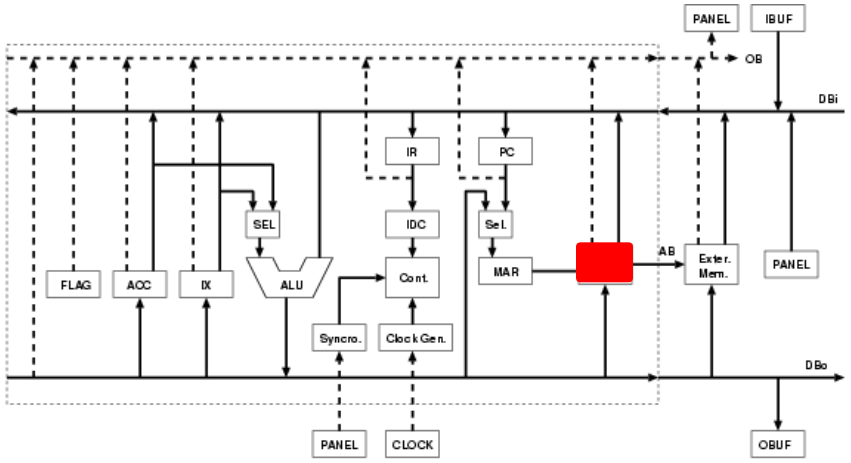
- メモリアドレスレジスタ
- メモリ操作の対象とするアドレスを保持.
- 8ビット



KUE-CHIP2: Internal memory (内部メモリ)

- 512バイト．バイト単位の番地指定
- プログラム領域：0～255番地
- データ領域：256番地～511番地





p.23 Fig. 3

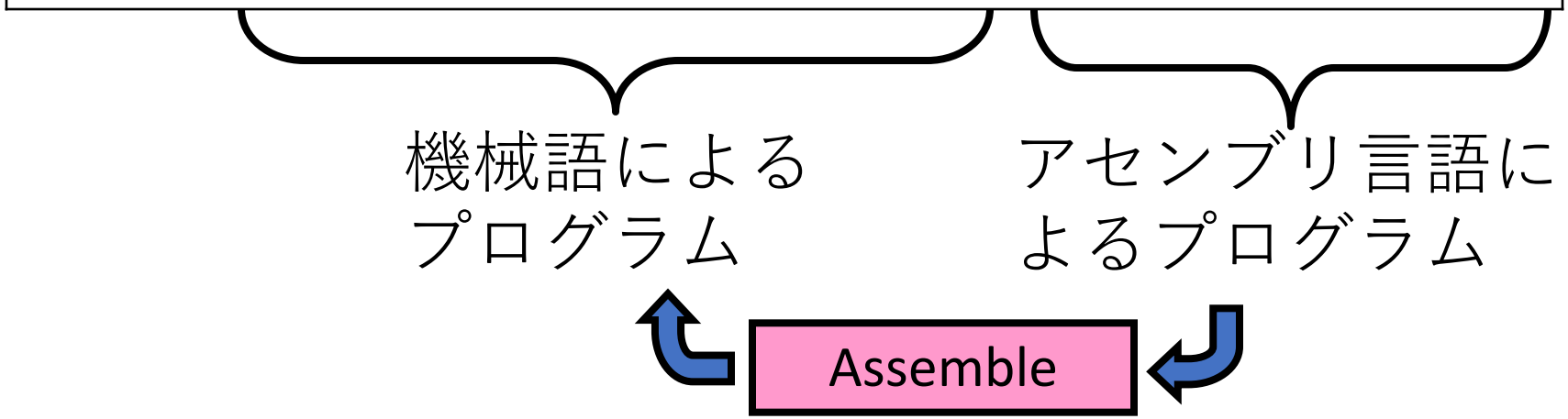
KUE-CHIP2のアセンブリ言語

- 命令の種類：p.24 表1を参照
- 言語仕様： p.35～38 付録A を参照

- 機械語フォーマット：1バイトか2バイト（p.23 図4を参照）

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

「01」という値をACCに格納する
「h」は16進数（hexadecimal）を表す

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

ACCの内容を出力バッファ(OBUF)に出力する

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

ACCの内容を論理左回転し，ACCに入れる

00000001

00000010

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

常に02番地へ戻る

How to assemble (1/4)

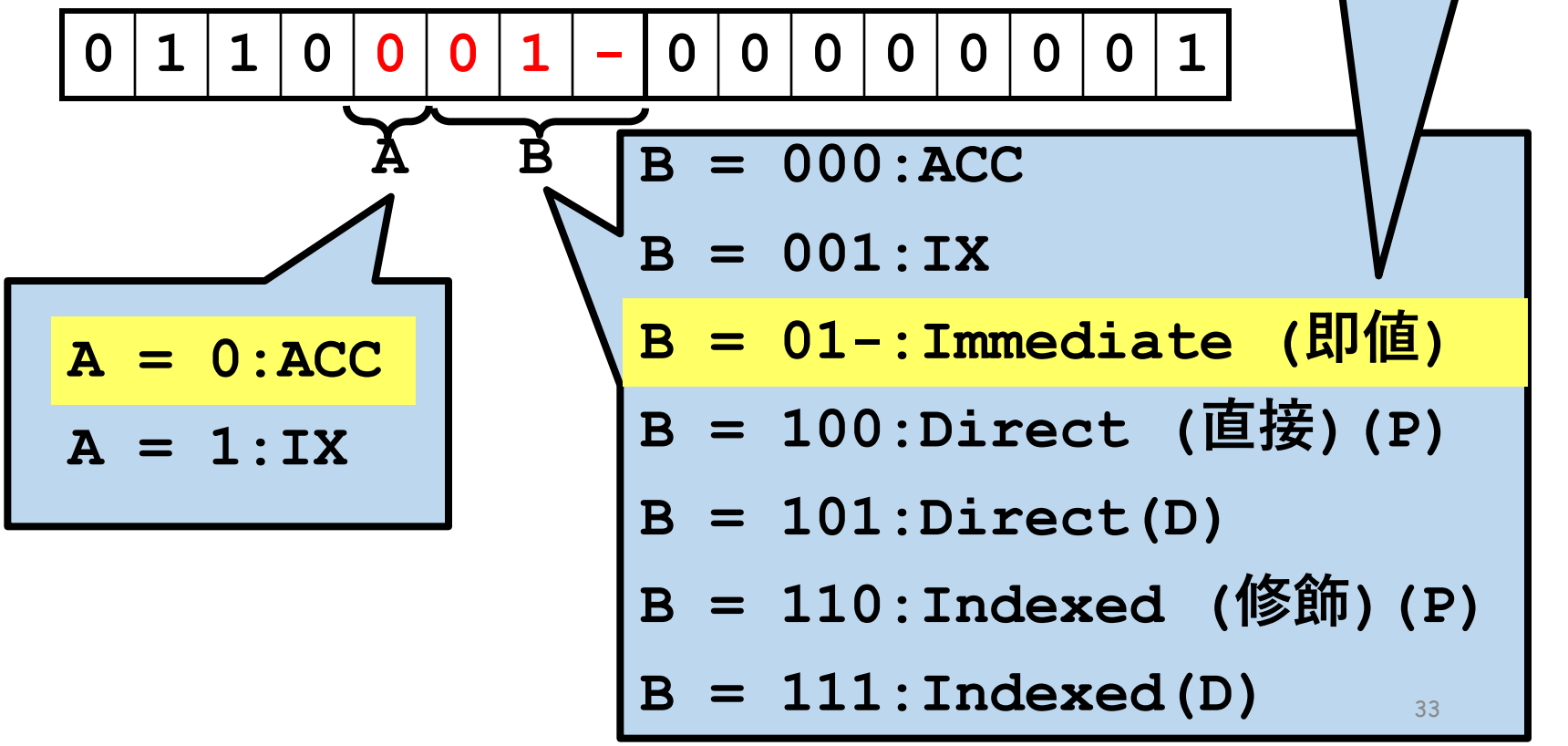
- Command table (p.37, Table 8)
- Assembly "**LD ACC,01h**"

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**"



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)

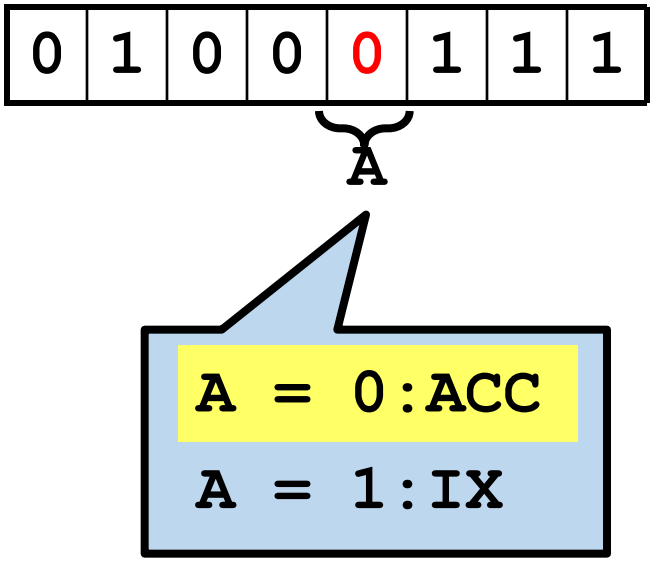
- Command table (p.37, Table 8)
- Assembly **"RLL ACC"**

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

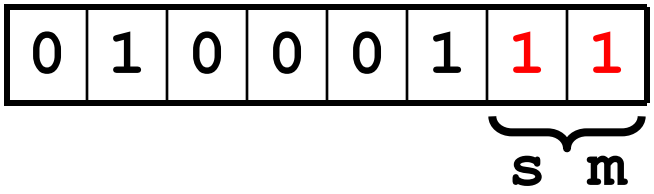
How to assembly (3/4)

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



How to assembly (3/4)

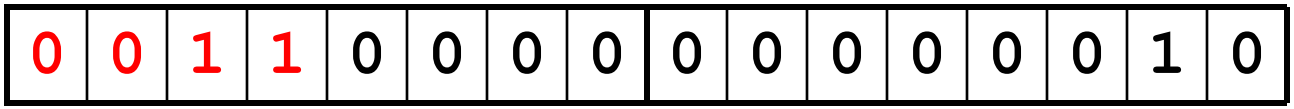
- Command table (p.37, Table 8)
- Assembly **"RLL ACC"**



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)

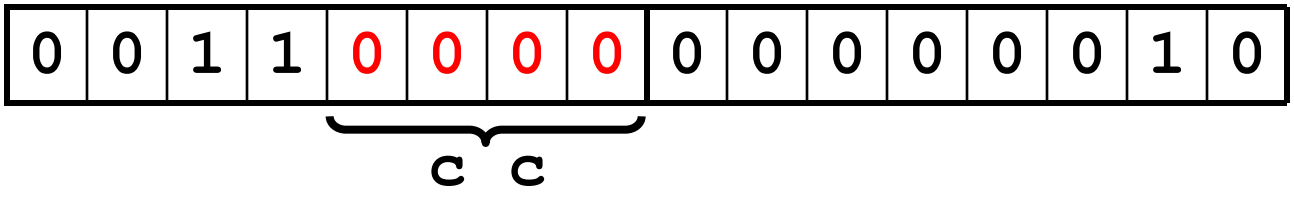
- Command table (p.37, Table 8)
- Assembly "**BA 02h**"



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**"



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

“–” は “do not care” を表す.
0か1で置き換える (どちらでもよい)

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

今日やること

- 導入
- KUC-CHIP2の基本的な使い方
- Problem 3.1
  - ADDとADCを実行しながら，ACC, PC, FLAGなどの値を記録する.
- Problem 3.3 (1)
  - クロック周波数を記録する
  - できるだけ440 Hzに近い単音を出力する
- 次の課題の説明

Execution of programs

- 第2.5節に沿って行う (p.26～32)
- 注意点:
  - 電源器とボードを接続してからコンセントに繋ぐこと
  - コンセントは机に固定されたものに繋ぐこと(転落防止)
  - 電源スイッチ横のコンデンサに指をかけないこと
  - プログラムの実行前にRESETを押すこと
- 全員確認できたら次の説明へ

操作方法の補足

- SSスイッチで実行
- さらにSSスイッチを押すと停止，再開
- CLKFRQのダイヤルを回すと実行速度が変化

- RESET
- SELスイッチを操作してACCを表示
- SIスイッチでステップ実行 (1命令ずつ)
- SELスイッチを操作してPCを表示
- SIスイッチでステップ実行 (1命令ずつ)

今日やること

- 導入
- KUC-CHIP2の基本的な使い方
- Problem 3.1
  - ADDとADCを実行しながら，ACC, PC, FLAGなどの値を記録する.
- Problem 3.3 (1)
  - クロック周波数を記録する
  - できるだけ440 Hzに近い単音を出力する
- 次の課題の説明

命令はどのように実行されるか?

- クロックに沿って実行
- クロック1周期分 → 1つの実行フェーズ

- KUE-CHIP2の各命令は3から5フェーズ
  - P0, P1：各命令で共通
  - P2以降：各命令で異なる

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」を見たら「80h」だと思う
(変数宣言，初期化のようなもの)

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			

メモリのプログラム領域D1番地の内容をACCに格納する

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			

メモリのプログラム領域D2番地の内容とACCの内容を加算する

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			

メモリのプログラム領域ANS番地にACCの内容を格納する

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			

プログラムの実行を停止する

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

メモリ80番地の内容を03とし，
81番地の内容をFD（-3）とする

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			

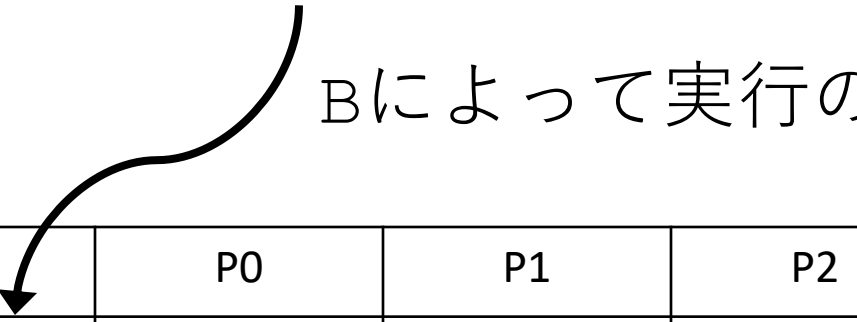
アセンブル結果の16進表示

Trace of the execution

LD ACC,[D1]

A **B**

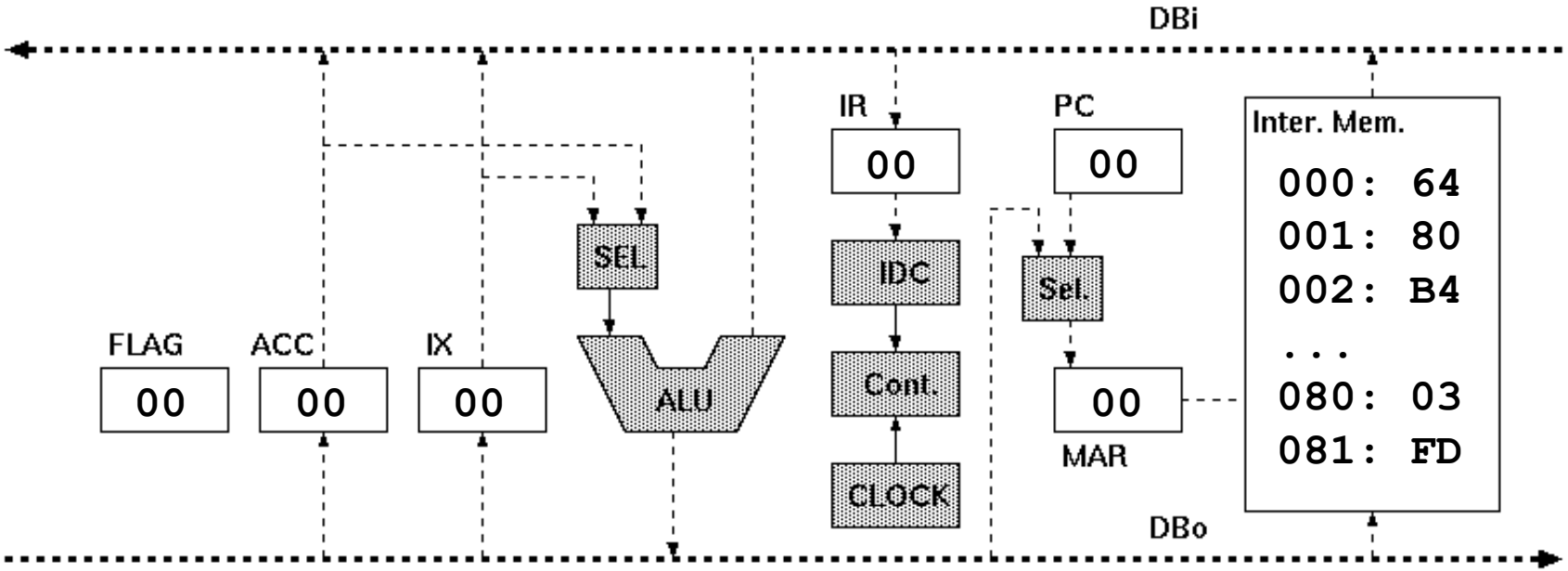
p.25 Table 2



		P0	P1	P2	P3	P4
LD	ACC IX	(PC)→MAR PC++	(Mem)→IR	(A)→B		
	d			(PC)→MAR PC++	(Mem)→A	
	[d] (d)				(Mem)→MAR	(Mem)→A

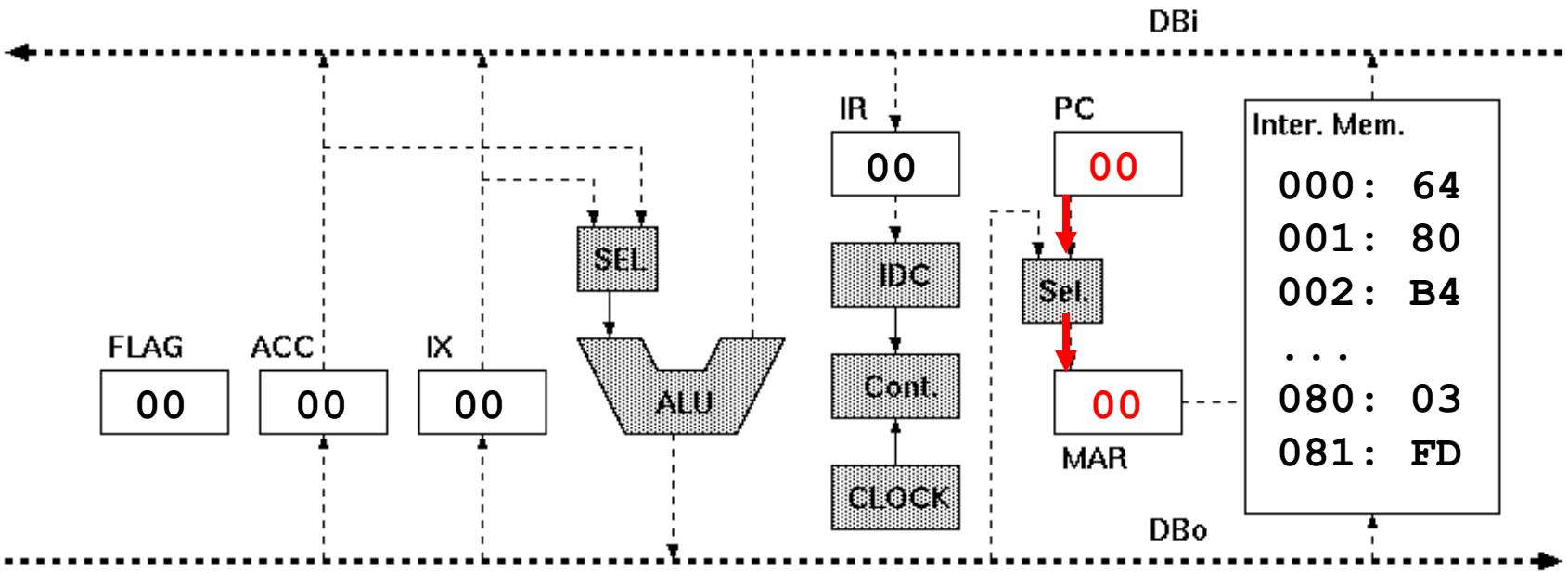
Trace of the execution

`LD ACC,[D1]`



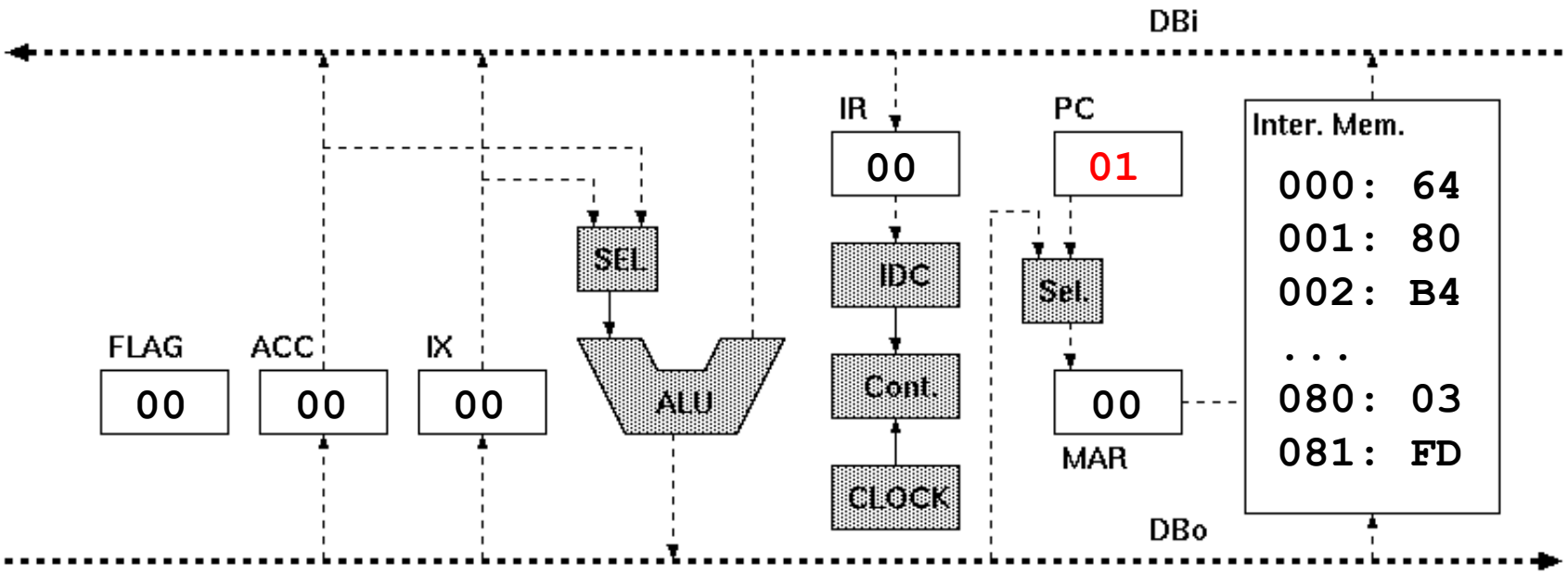
Trace of the execution

LD ACC,[D1] P0: (PC)→MAR, PC++



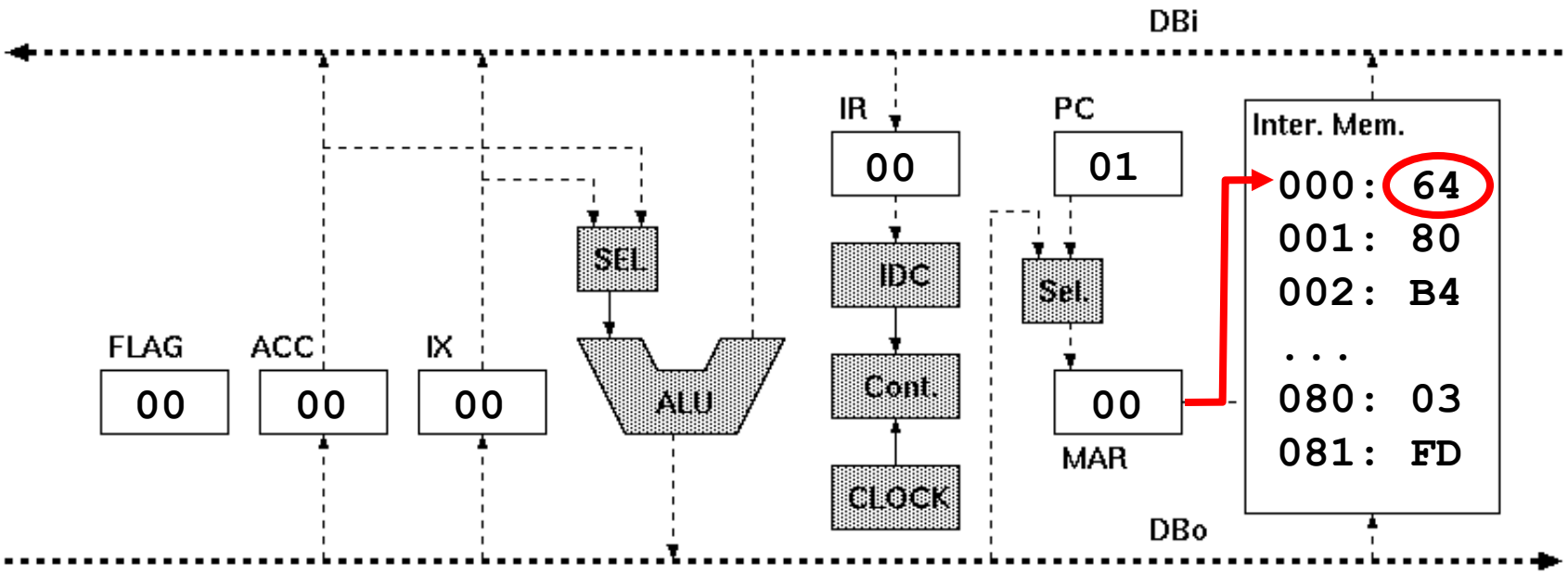
Trace of the execution

LD ACC,[D1] P0: (PC)→MAR, PC++



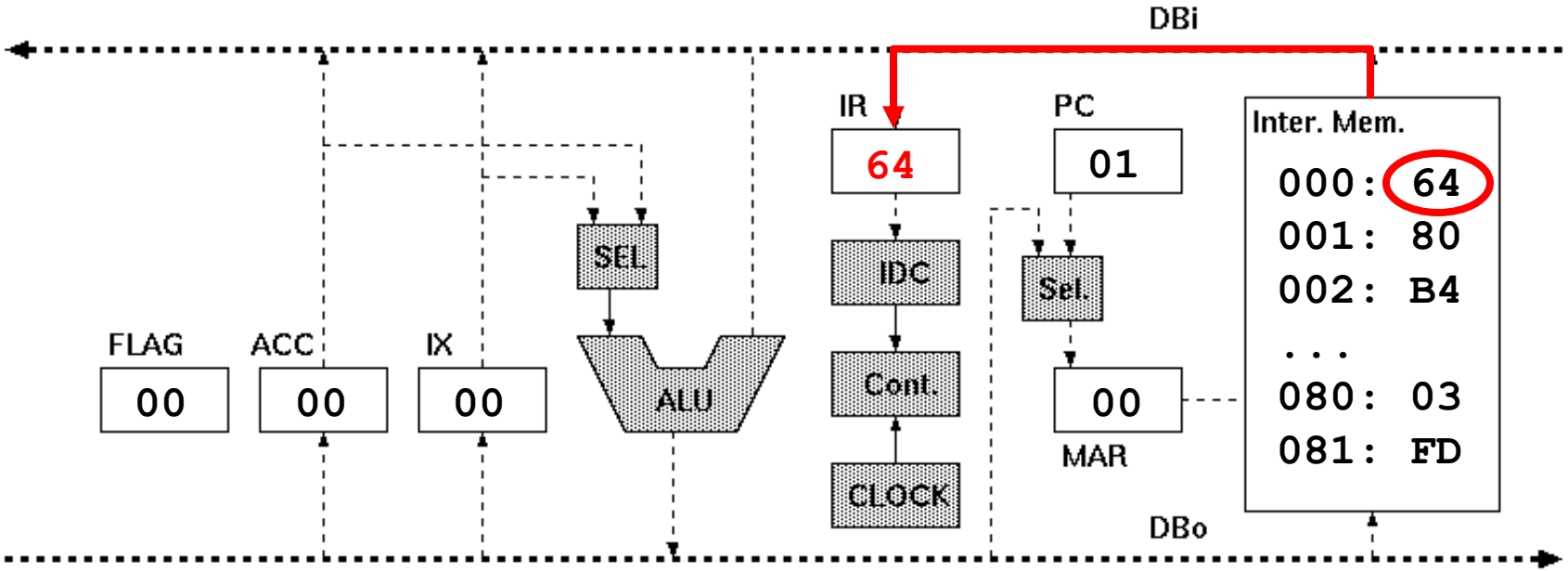
Trace of the execution

LD ACC,[D1] **P1: (Mem)→IR**



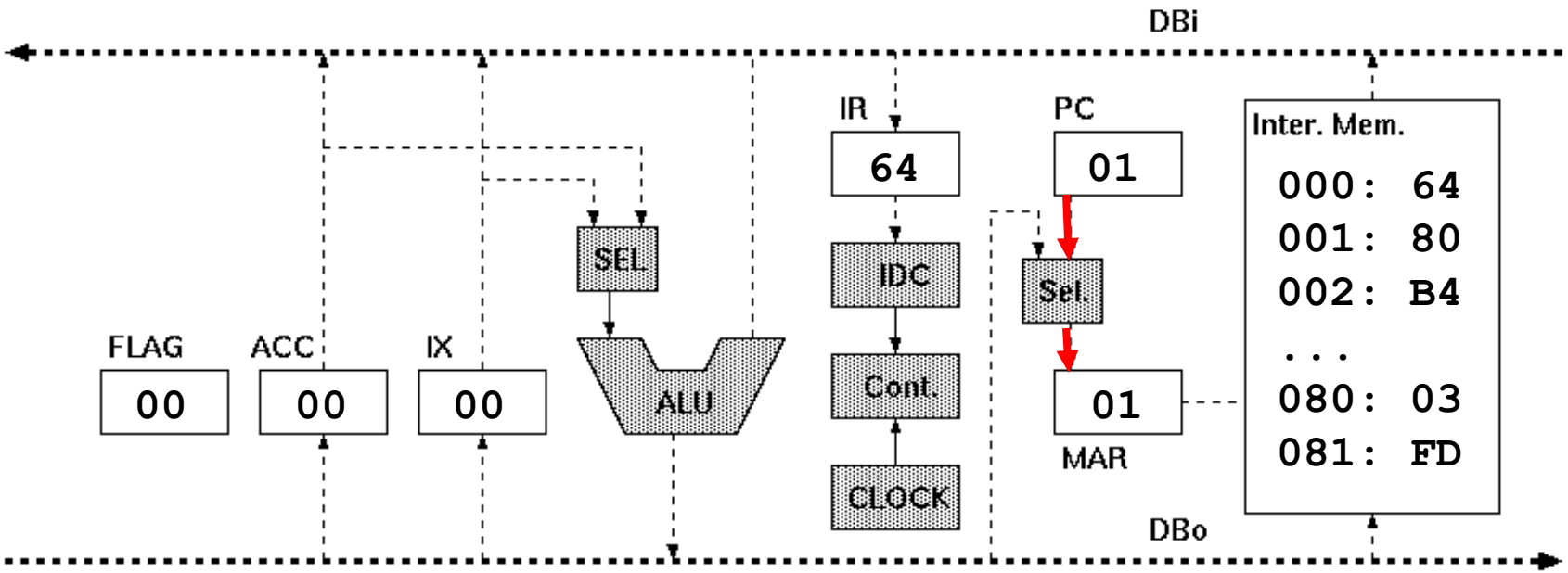
Trace of the execution

LD ACC,[D1] **P1: (Mem)→IR**



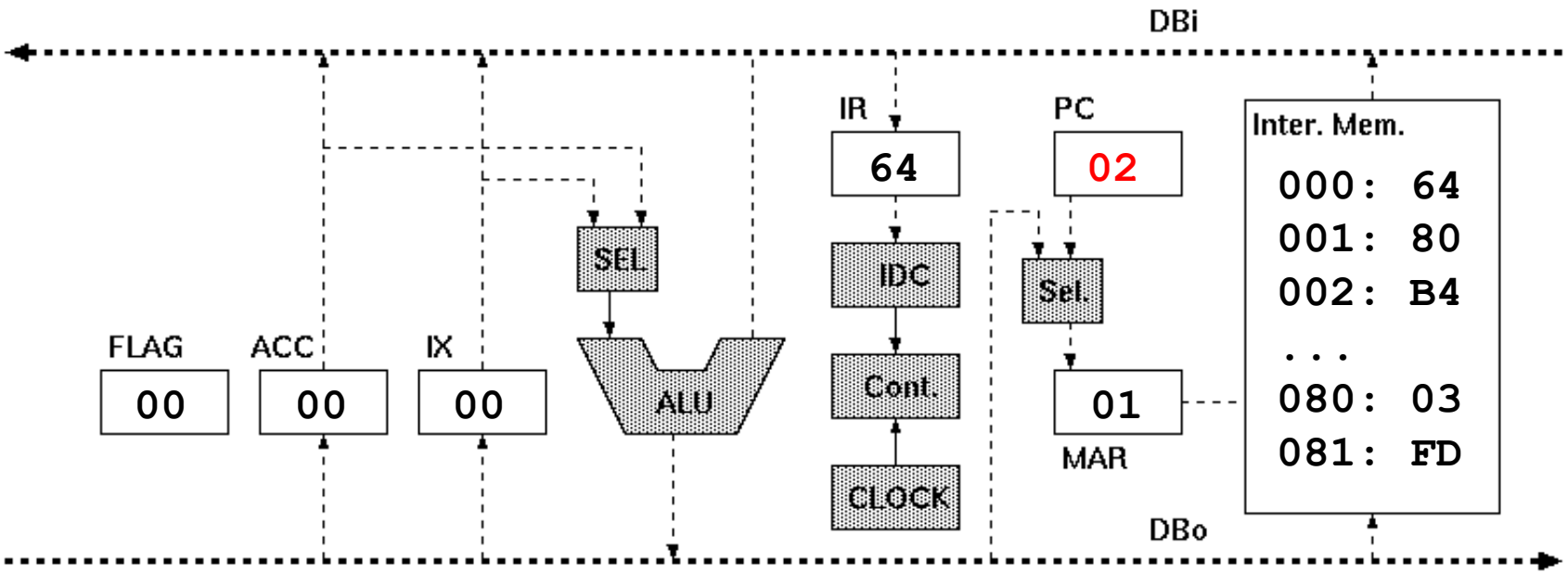
Trace of the execution

LD ACC,[D1] P2: (PC)→MAR, PC++



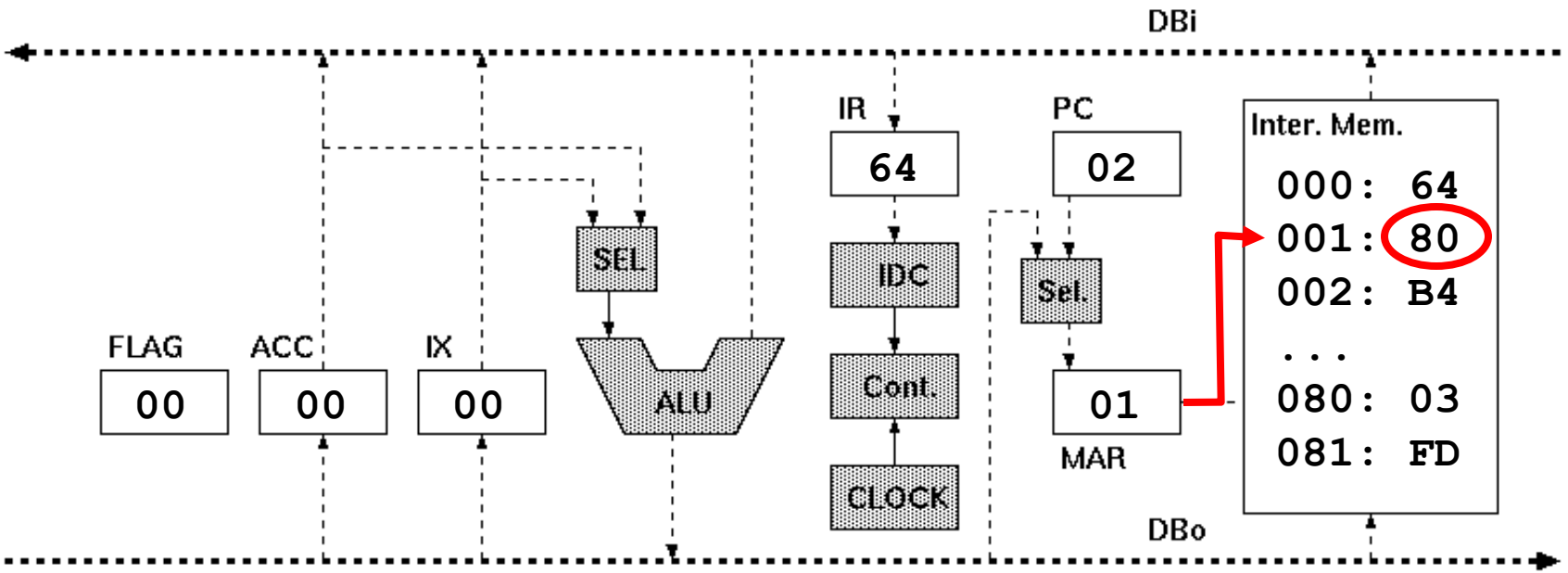
Trace of the execution

LD ACC,[D1] P2: (PC)→MAR, PC++



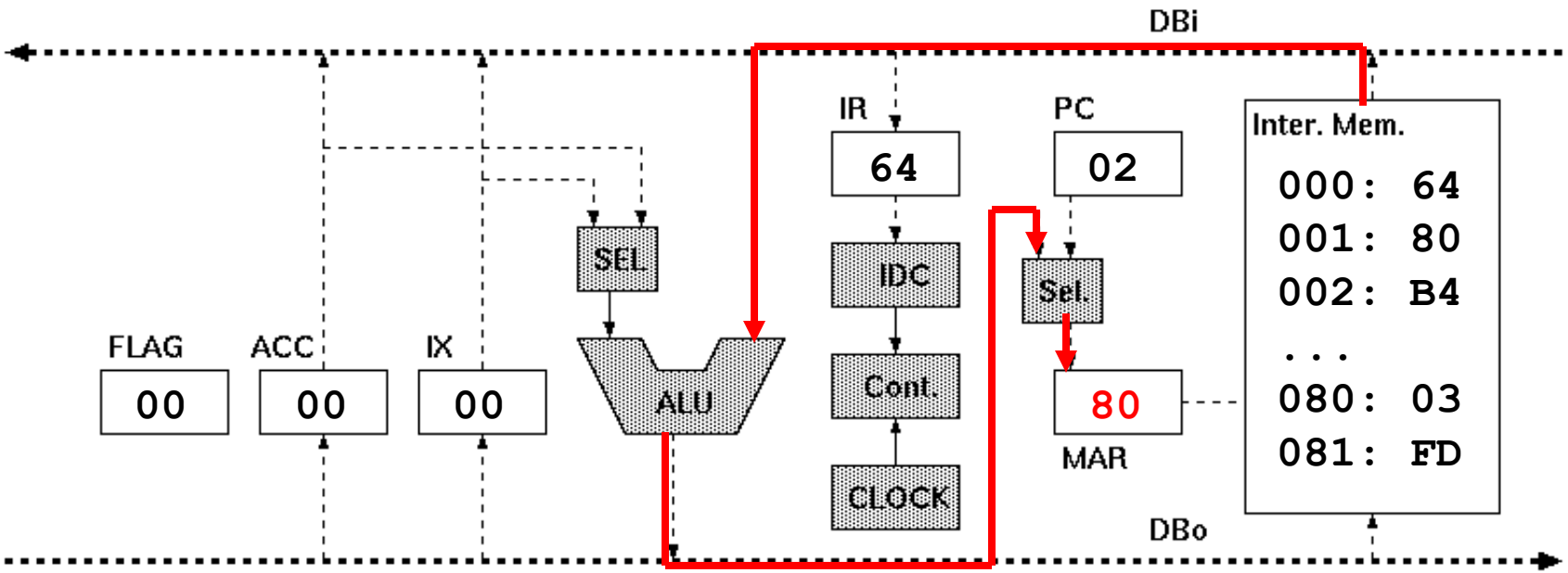
Trace of the execution

LD ACC,[D1] P3: (Mem)→MAR



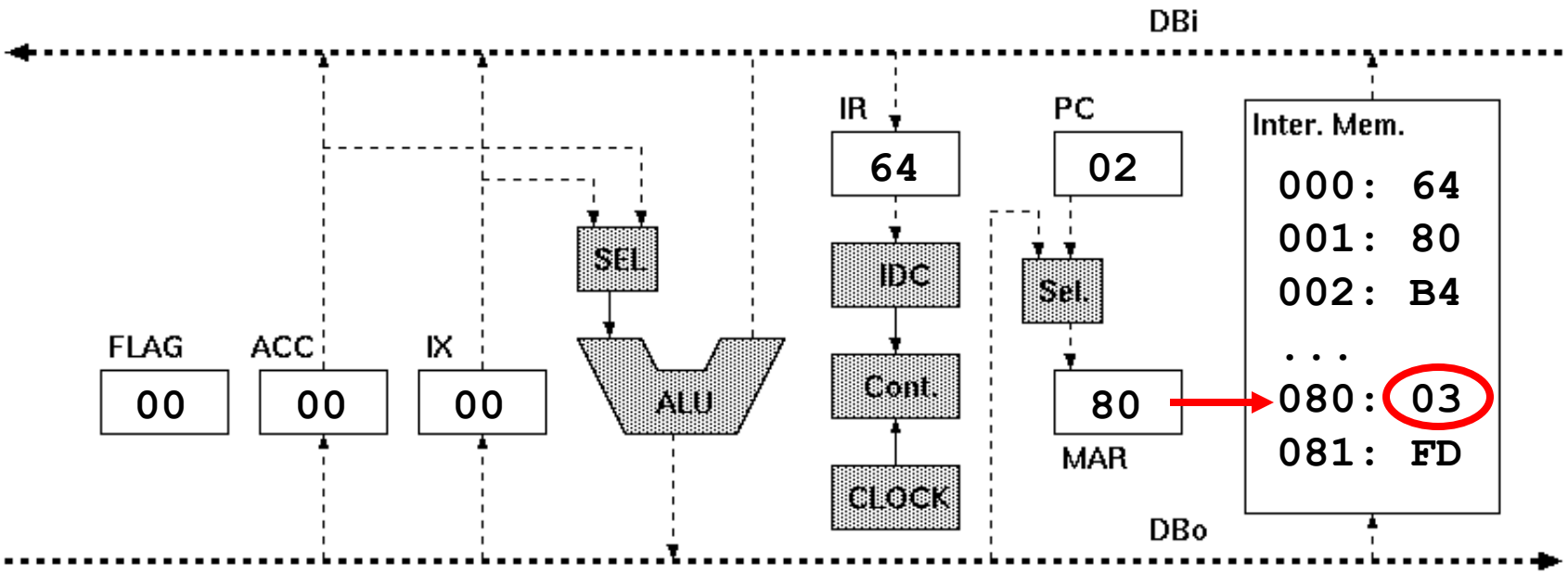
Trace of the execution

`LD ACC,[D1]` **`P3: (Mem)→MAR`**



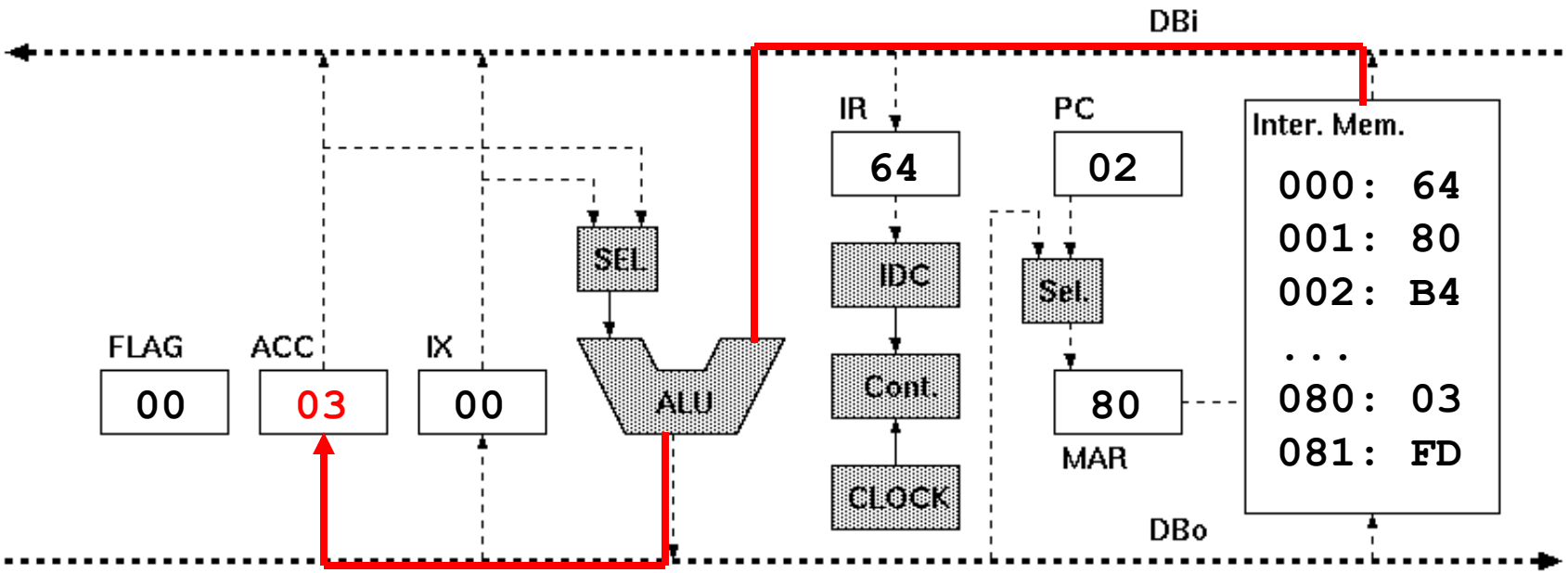
Trace of the execution

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)
  - 演算結果に桁上がりが生じるとCF = 1.
- Overflow Flag, VF (桁あふれフラグ)
  - 演算結果に桁あふれが生じるとVF = 1.
- Negative Flag, NF (負フラグ)
  - 演算結果が負になると NF = 1
- Zero Flag, ZF (ゼロフラグ)
  - 演算結果がゼロになると ZF = 1.

p.22 Fig. 2

Problem 3.1 (p.33)

- (1)
  - 実行開始から実行終了まで，観測可能なレジスタ，バスをトレース
- (2)--(6)
  - ADD開始前からADD終了後まで，フラグレジスタのみをトレース
  - ADD命令をADC命令に変更して，ADC開始前からADC終了後まで，フラグレジスタのみをトレース

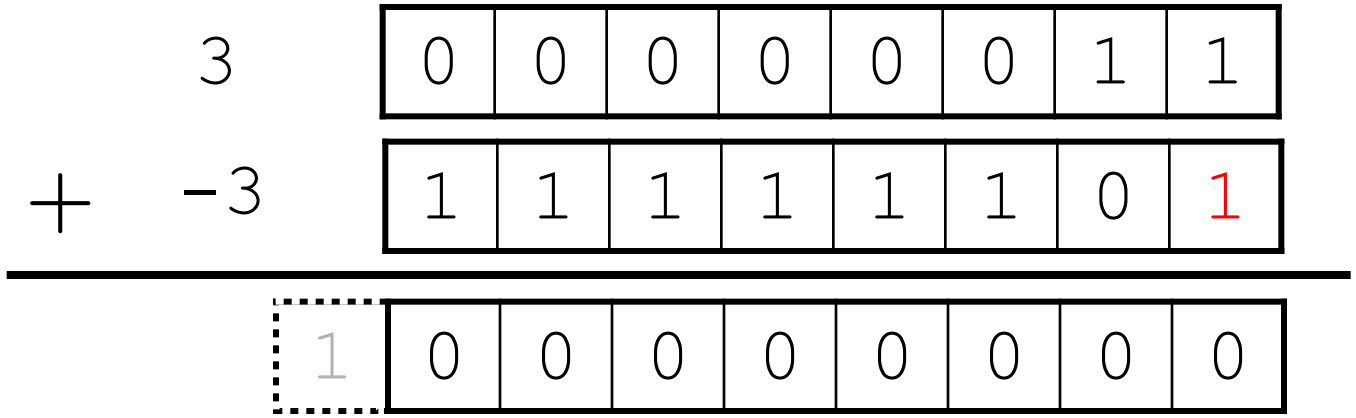
  - それぞれの加算結果も確認・記録すること

Problem 3.1: Caution 1/2

- 16進数「64」，2進数では？
- 80番地に値を入れるには，まずMARを操作
- 毎回，まず計算結果を確認(記録)すること
- 「6」と「b」の読み間違いに注意

Problem 3.1: Caution 2/2

- 負の数は「2の補数表現」



Points for report

- (1)
  - 各命令の各フェーズでの動作についてテキスト p.24〜28 を参考に図などを使いながら文章で説明すること
    - レポート作成補助：図や資料のデータを配布中
    - https://expcs.github.io/microprocessor/
- (2)--(6)
  - 各フラグがどのような時に変化するのか，ADD命令とADC命令の違いもまとめること

今日やること

- 導入
- KUC-CHIP2の基本的な使い方
- Problem 3.1
  - ADDとADCを実行しながら，ACC, PC, FLAGなどの値を記録する.
- Problem 3.3 (1)
  - クロック周波数を記録する
  - できるだけ440 Hzに近い単音を出力する
- 次の課題の説明

Output a melody

- Output waves from KUE-CHIP2 to generate a sound from a speaker.
- KUE-CHIP2から波を出力し，スピーカから音を出す

- 今日: 音を出す仕組みの基礎を学び，単音を出力する
- 3周目: メロディー出力プログラムの実行

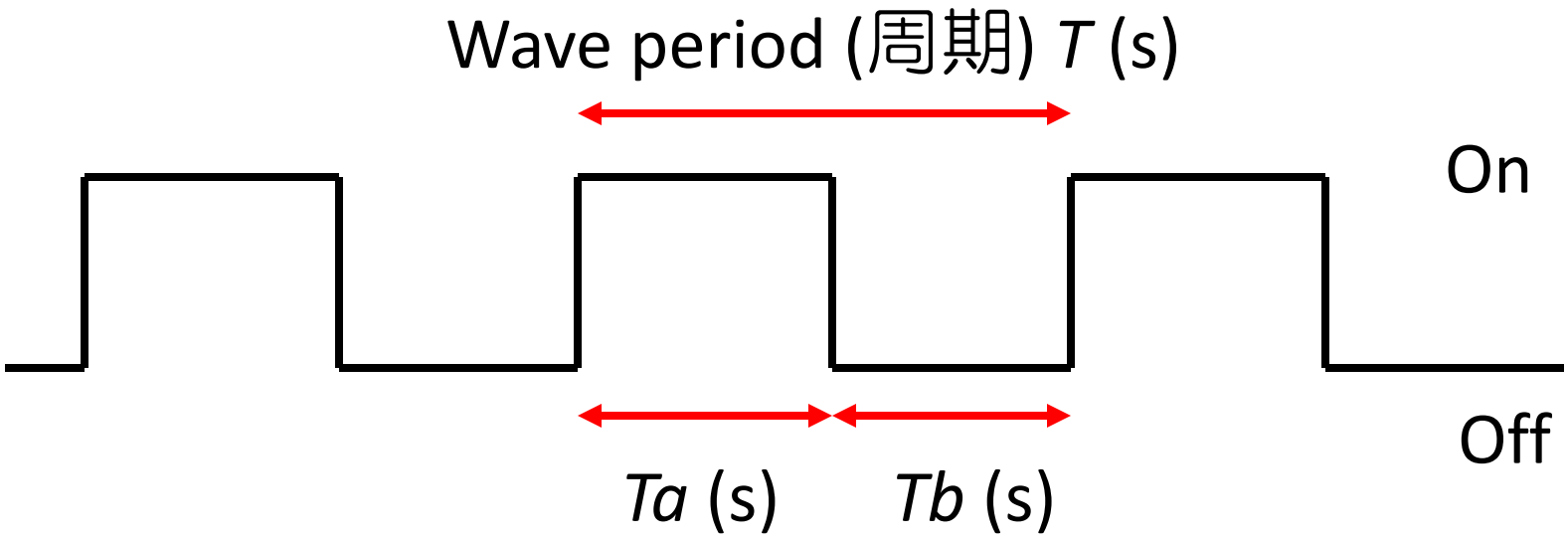
What is sound?

- 音は空気の振動 (波)
- 音の三要素
  - 大きさ: 波の振幅の大きさ
  - 高さ: 波の周波数の高さ
  - 音色: 波の形

- スピーカ:
  電気信号を音（空気振動）に変換する装置

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

自分で決める

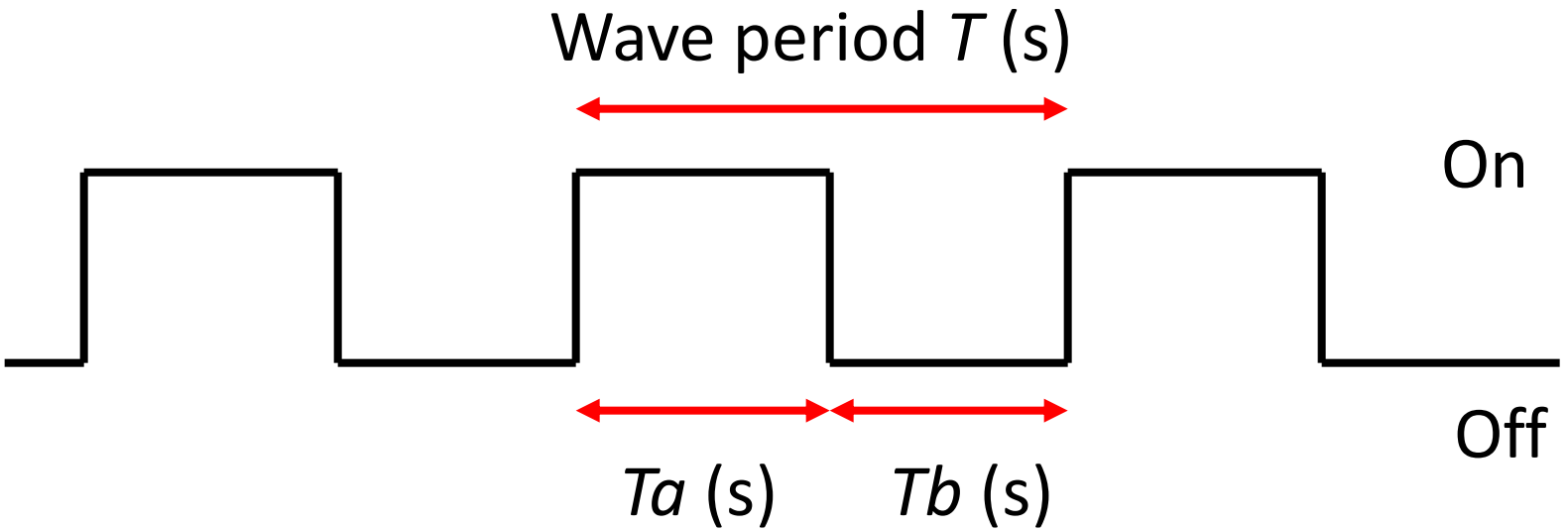
自分で決める

Wave generation (p.39, List 4)

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

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) オシロスコープでクロック周期を確認
  - スイッチCLKを中立に
  - ダイヤルCLKFRQの「0～8」の周波数を測定
  - 信号はJP3（右列の上から2番目）より出力
- (b) リスト4のa, bを設定する
  - 出力する音の周波数：440 Hz 「ラ」
  - 最適なT0, a, bを計算によって定める
    - T = Ta + Tb, T = 1/440 (s)
    - Ta ＝ (12+8a)T0, Tb ＝ (16+8b)T0

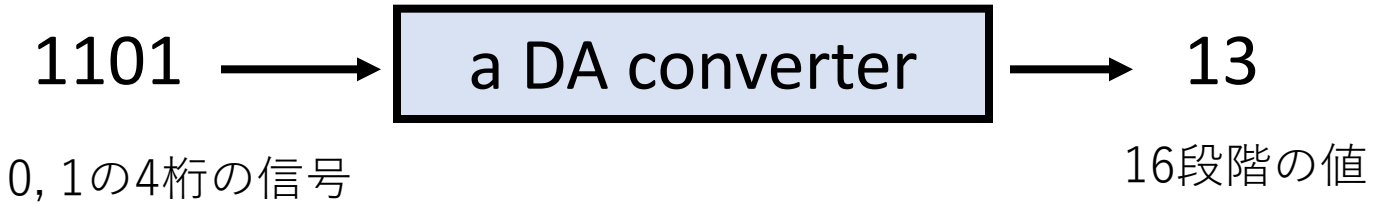
Problem 3.3 (1) p.33

- (c) 440 Hzの音の出力
  - リスト4の入力
  - CLKFRQの設定
  - DAコンバータを通して，オシロスコープで周波数を確認
  - 出力音が440 Hz (誤差±1%) であることを
  - 計算によって確認

Digital to analogue value

- 出力バッファにDAコンバータを付けて出力信号をオシロスコープへ

- DAコンバータ (DAC):
  - ディジタル信号をアナログ信号に変換する回路



DACに関する注意

- 使用するDAコンバータはとても壊れやすいので，大事に扱うこと（むやみに触らない）
- 特に，取り付け部分周辺の配線に注意
- 取り付け＆取り外しは教員・TAが行います

繋ぎ方

- Connect the DAC to the oscilloscope;
  channel 1 → Red
  channel 2 → Blue
  ground → Black
- ダイヤルCLKFRQを「1」にして実行

Notes for your report for (3) メロディの出力 (a) 誤差±1%の確認

- どのように最適なT0, a, bを計算したか？
  - 計算過程を記述すること
- どのように確認を行ったか？
  - 実際に誤差を計算すること
- 他の精度確認方法は考えられるか

Notes for your report for (3) メロディ出力 (b) 精度をより上げるための対策

- KUE-CHIP2だけで対処する場合（ソフトウェア上，プログラム上の工夫）
- その他の機器をKUE-CHIP2に接続する場合（ハードウェア上の工夫）

※メロディー出力の基本的なアルゴリズムはそのまま出力周波数を440 Hzに近づける方法

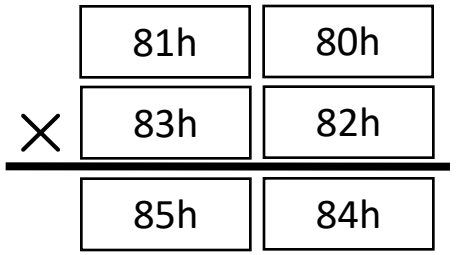
今日やること

- 導入
- KUC-CHIP2の基本的な使い方
- Problem 3.1
  - ADDとADCを実行しながら，ACC, PC, FLAGなどの値を記録する.
- Problem 3.3 (1)
  - クロック周波数を記録する
  - できるだけ440 Hzに近い単音を出力する
- 次の課題の説明

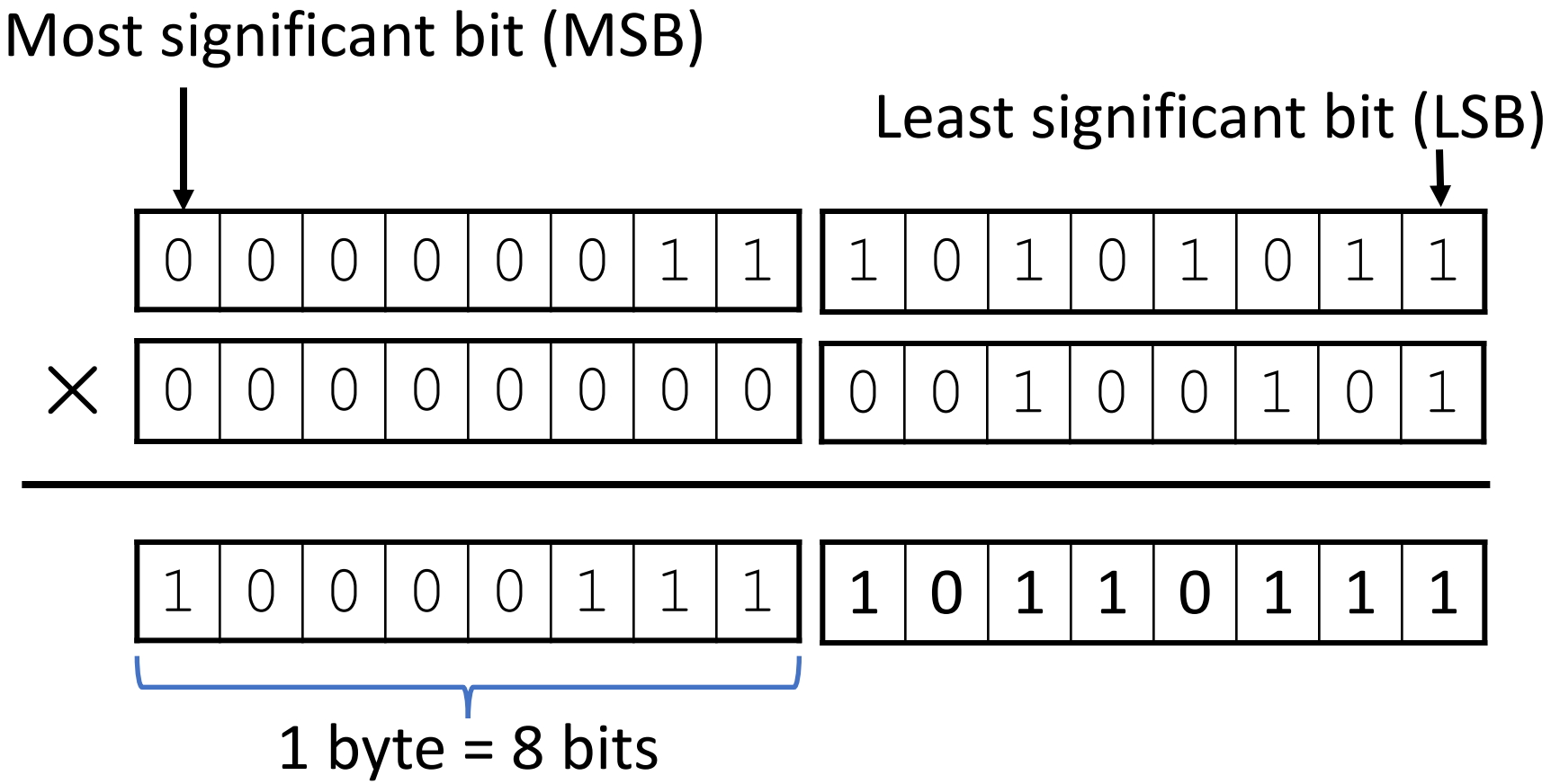
Next class: Problem 3.2: Multiplication

- 符号なし2バイト精度の2数の積
- データを格納する番地はテキスト通りでなくても良い
- 演算結果は2バイトに収まると仮定

- Preparations
  Prepare and assemble a program



補足：符号無し２バイトの乗算



メモリ上での順序に注意

補足：アドレスモード

- オペランド(引数)の表現方法のこと
- KUE-CHIP2のアドレスモード (p.29～31参照)
  - ACC,IX：ACC,IXの内容がデータ
  - 即値：オペランドそのものがデータ
  - 直接：オペランドがメモリのアドレス．そのアドレス上の内容がデータ
  - 修飾：「オペランド＋IXの内容」がメモリのアドレス．そのアドレス上の内容がデータ

補足：命令について (p.24)

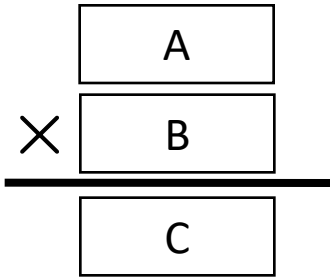
- ADD: 加算命令．CFを考慮しない
- ADC: 加算命令．CFを考慮する
- SUBとSBCも同様の関係.

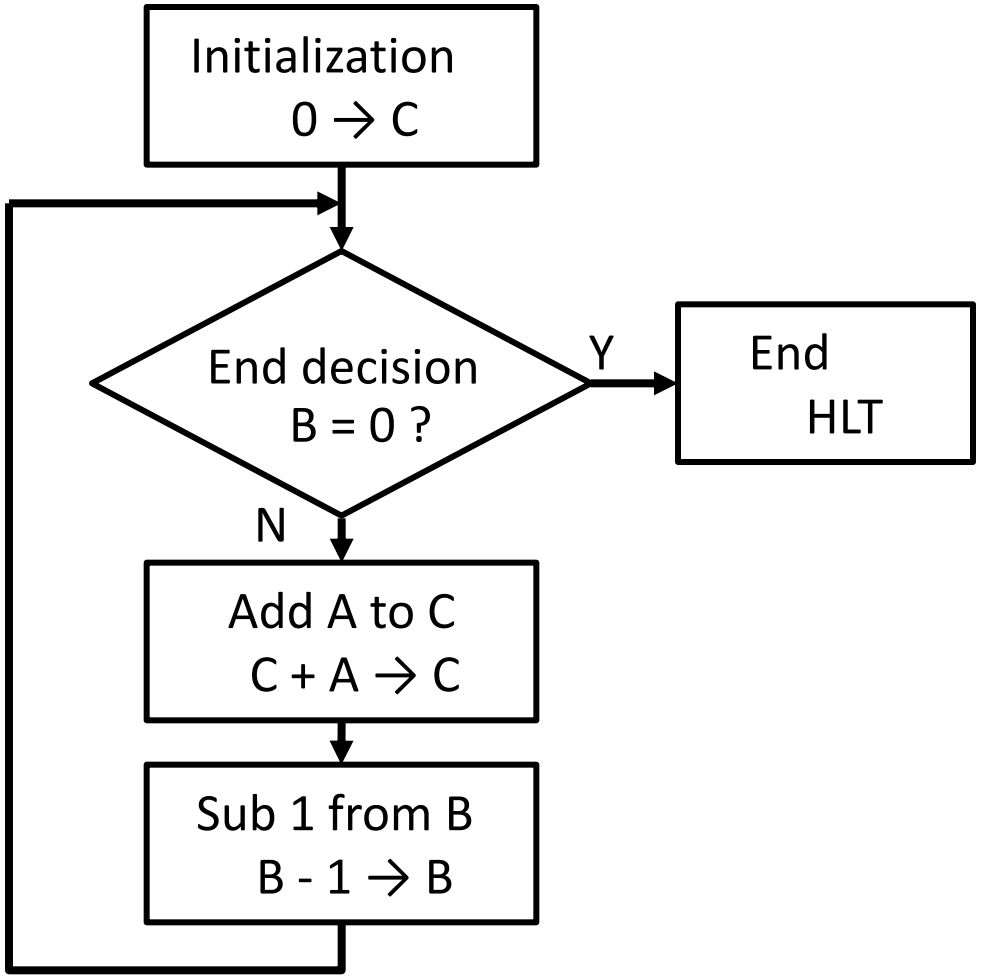
- RCF: CFをリセットする

Notes

- 必ずプログラムを準備してくること！
- まずはフローチャートを作成すること
  - 授業開始時(入力中)に問題がないか確認する
  - プログラムとフローチャートは別の紙に
- 遅刻や準備不足に対する救済は行わない
  - できなかった分は減点，最悪の場合は不受理

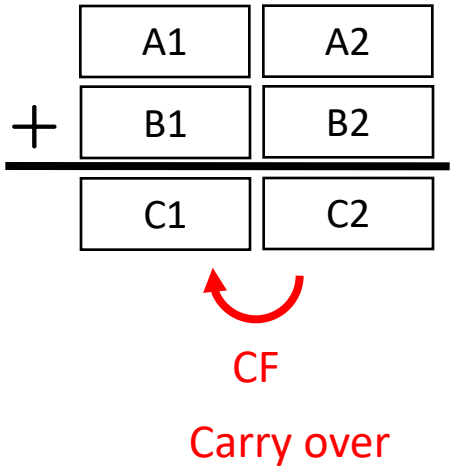
フローチャートの例
1バイトの乗算

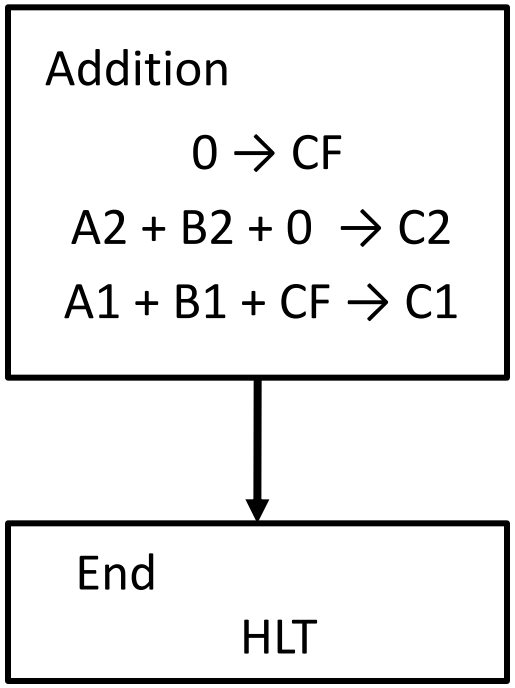




プログラムの流れを
自然言語で図示する

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

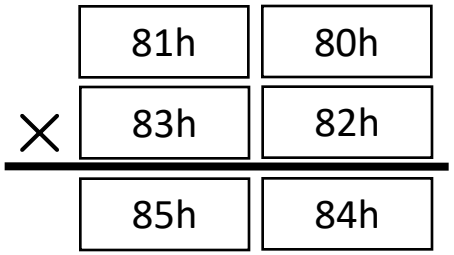
- 他人が見て分かるように書くこと
- 必ず紙に手書き or 印刷してくること
- 紙の両面を使わない
- アセンブリ言語と機械語は横に揃える
- 修正用のスペースも用意しておく
- 機械語は2進・16進のどちらでも良いが，16進数なら確認しやすい

アセンブリ言語と機械語は横に揃える

```
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

よくある間違い



- 2バイトのデータの取扱い
  - 上位・下位バイトの番地の誤り
- 初期化のし忘れ
  - `SUM += A`
- 繰り上げの失敗 (ADD, ADC, RCF)
- 終了判定の誤り
  - “`LD 0`” では`ZeroFlag`は立たない
- データの保存 (ST) のし忘れ
- アドレスが16進数ではなく10進数になっている

エミュレータを使った準備

- 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 (←おすすめ)
- https://emu.kemuide.openwaseda.net

今日やること

- 導入
- KUC-CHIP2の基本的な使い方
- Problem 3.1
  - ADDとADCを実行しながら，ACC, PC, FLAGなどの値を記録する.
- Problem 3.3 (1)
  - クロック周波数を記録する
  - できるだけ440 Hzに近い単音を出力する
- 次の課題の説明

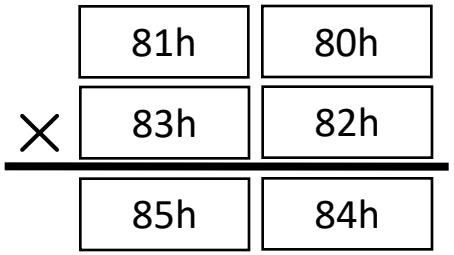
Microprocessors (Lecture 2)

Lecture 2

- Problem 3.2 乗算プログラムの作成
- 符号なし2バイト精度の2数の積
- データを格納する番地はテキスト通りでなくても良い
- 演算結果は2バイトに収まると仮定

- 必須の予習：
  プログラムの作成とアセンブル

よくある間違い



- 2バイトのデータの取扱い
  - 上位・下位バイトの番地の誤り
- 初期化のし忘れ
  - `SUM += A`
- 繰り上げの失敗 (ADD, ADC, RCF)
- 終了判定の誤り
  - “`LD 0`”では`ZeroFlag`は立たない
- データの保存 (ST) のし忘れ
- アドレスが16進数ではなく10進数になっている
- 入力ミス，アセンブルの誤り

Procedure

- 各自の作成したプログラムを入力
  - 入力中にフローチャートをチェック
- ホワイトボードの(1)～(4)で動作確認
- それらが正しく計算できたら (A)と(B)を計算. 実行時間を計測（100 Hzで）
- ホワイトボードに実行時間とプログラムのメモリ消費量（単位：バイト）を記入
  メモリ消費量 = プログラム部分 + データ格納部分

実行時間の理論値

- 自分のプログラムについて実行時間の理論値を求め，実測値と比較せよ
  - 手順1) 実行時間を決めるパラメータを特定
    - 各命令のフェーズ数 (p.18表2)
    - 1フェーズ ＝ 1クロック周期
    - クロック周波数 ＝ 100 Hz
  - 手順2) 実行時間を求める計算式を導出
  - 手順3) 式から(A),(B)の実行時間理論値を算出
  - 手順4) 理論値と実測値の比較

Notes for your report

- 使用したプログラムのリストを載せ，フローチャートを用いて説明せよ
- 他の人（最低2人）のプログラムと比較
  - 論点１：実行時間 (実測値で可)
  - 論点２：プログラムのメモリ消費量

- 注意：他の人のプログラムは掲載不要だが，簡単な説明は記述すること

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

- 必須の予習：
  プログラムの作成とアセンブル
- 参考：Appendix B.2 and list 5 (p.41)
  - 楽譜データを用意するだけではダメ
  - List 5のプログラムに改造が必要
- 時間内に完成しなかった場合は打ち切りデバッグのサポートはできるが，プログラムが無い場合はサポートできない

Notes

- メロディーの出力は無限に繰り返すこと
- 最も高周波・低周波な音でも可聴領域を超えない
- p.40 表13「音階の周波数」を参考に
  - 1オクターブ高い音 → 周波数が2倍
- リスト5に改造が必要な部分
  - 「休符」はどうすれば実現できるか
    → 音符と休符を判別し，別処理が必要
  - 同じ音が続くと1つの長い音に聞こえる
    → 音と音の間に空白が必要

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)
```

00, 1A, 1C, 18はデータ領域のアドレスを指している (自分のプログラムに合わせて設定)

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
```

音の先頭

音の終わり

どの音を鳴らすか (3ずつ増える)

Output

実行時に使用

n1は音の高さ，n2 n3は長さ (2重ループ)

Microprocessors (Lecture 3)

Problem 3.3 (2) Output a melody

- 簡単なメロディーを出力させる
- 必須の予習：
  プログラムの作成とアセンブル
- Reference: Appendix B.2 and list 5 (p.41)

- DAコンバータの取扱いに注意

Notes for your report
(4)(c) 各自のデータ表現の特徴をプログラムを載せて説明せよ

- 例えば，「楽譜」は人間に理解しやすいようにメロディーを表現している
- どのような表現なら理解しやすいのか

Notes for your report
(4)(d) 作成したメロディー出力法は他のCPUにも流用できるか？

- 他のCPUの例を1つ挙げて考察
- 挙げたCPUの実行命令フェーズを調べ，それを踏まえて考察

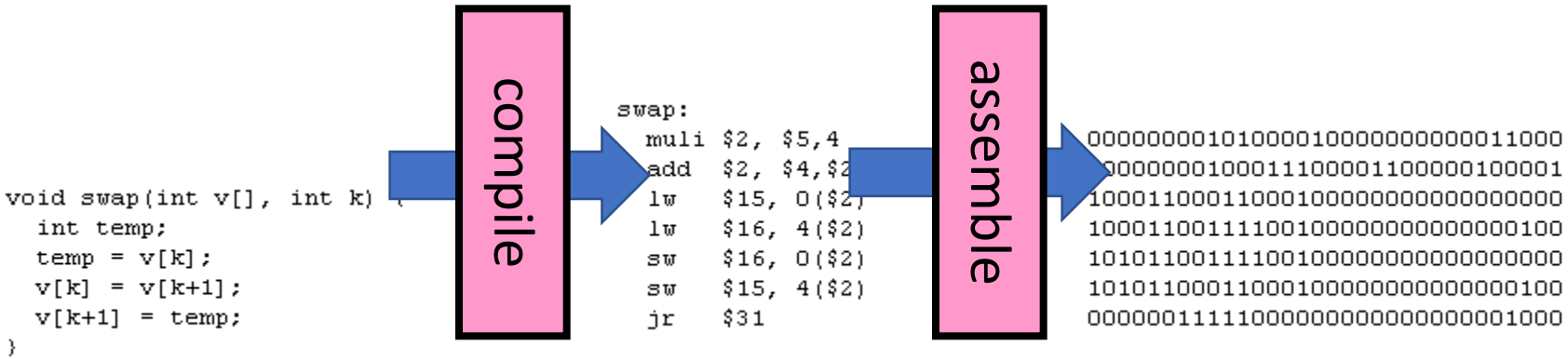
Notes for your report
(5)) 自分が最も使用しているCPU (または，有名なCPU) について，そのアーキテクチャを調べてまとめる

- レジスタ，命令セット，メモリ空間の特徴
- 乗算命令がどのように実行されているか
- 任意の課題（必須ではない）
  - やらなくても良いが，この課題は加点対象

Summary

- 計算機の仕組みについて理解
  - 例えば，なぜ32bitのOSでは4G以上のメモリが使えないのか？
- プログラムの作り方やデバッグの練習

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



高級言語によるプログラム　　アセンブリ言語によるプログラム　　機械語によるプログラム

Report submission 1/3

- 指導書p.6をよく読むこと
- PDFファイルをメール（fukumura@cs.tut.ac.jp）で提出
- 表紙は自作のものでも構わない
- 実験方法について，指導書を丸写しする必要はない
- この資料の「検討事項のポイント」を参考に
- 必ず自己点検票をチェック (提出は不要)

Report submission 2/3

- 提出〆切は1週間後の23:59 (時間厳守)
  - 病気等の例外を除き，〆切の延長はしない
  - 受理されたものへの改善・修正は可 (一週間以内)
  - 未完成のもの(途中までしかないもの)は不受理

Report submission 3/3

- メールの件名:
  - [report] –[student ID]–[your name]
  - [レポート]-B123456-豊橋太郎
- レポートはPDFに変換
- 添付ファイル名は :
  - [Your school register number]-[your name].pdf.
  - B123456-豊橋太郎.pdf
- 3日たっても確認メールが届かない場合は, F-408へ
- 分からないことがあれば何でも質問すること