2023後期教養教育　共通科目「学問と社会」 20223. 11.14. 8:45~10:15; 10:30~12:00

コンピュータの素顔とアルゴリズムデザイン

広島大学教育本部 客員教授
広島大学 名誉教授
渡邉敏正

概要：コンピュータの仕組み、文字や数値の扱い、
アルゴリズムデザイン、プログラムによって
コンピュータを直接的に操る様子などを
分かりやすく解説します

講義概要

1. コンピュータの素顔は？
   （イントロダクションに代えて）
2. コンピュータの機能（ハードウェア）
   と情報処理の仕組み（ソフトウェア）
   講義前半のまとめ
3. 情報の収納方法（データ構造）と
   処理手順（アルゴリズム）
   講義後半のまとめ

1. コンピュータの素顔は？（イントロダクションに代えて）

1.1 コンピュータは魔法使い？
1.2 コンピュータは台所とお母さん
1.3 現代社会におけるコンピュータの役割
1.4 コンピュータの素顔

1.1 コンピュータは魔法使い？

- ワードでレポートを書く
- エクセルで表計算をする
- パワーポイントでプレゼンをする

- ブラウザでインターネットの情報を集める
- アウトルックで電子メールを送る

1.2 コンピュータは台所とお母さん(1)

調理
（まな板、
包丁、コンロ、
・・・）

制御装置

演算装置

考える
（指示）
作る
（二役）

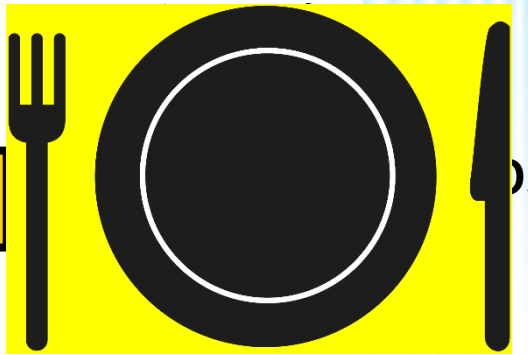
入力装置

出力装置

キーボ　ス
ディス　、

食材の買物

食材、料理、
レシピの保存



食卓などへ

デー　　制御関係

1.2 コンピュータは台所とお母さん(2)

調理
（まな板、
包丁、コンロ、
・・・）

制御装置

演算装置

考える
（指示）
作る
（二役）

入力装置

キーボ　ス
ディス　、

食材の買物

食材、料理、
レシピの保存

食卓などへ

デー　れ

制御関係

1.3 現代社会でのコンピュータの役割(1)

- Microsoft Office 、
- ホームページおよび
- ）

- 航空管制システム、
- 冷蔵庫、コピー機、エアコン、ポット、
- ロケット、人工衛星、航空機、船舶、自動車、列車、

1.3 現代社会でのコンピュータの役割(2)

1.4 コンピュータの素顔(1)

電子部品の詰まった箱＋基本ソフトウェア（OS）

＝ コンピュータ

電子部品の詰まった箱

トウェア（OS）
s, MacOS など）

から動作の指示を受ける
に文字や画像を表示する
ックで動作の指示を受ける
ョンソフトを起動する

実行

メモリ内に格納

実行

ケーションソフトウェア

- Office
- 各種ブラウザ
- その他

1.4 コンピュータの素顔(2)

アプリケーションソフトウェア

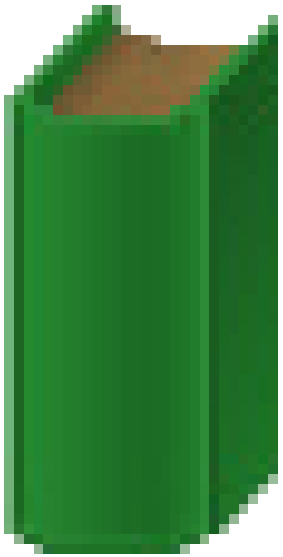
- Microsoft Office 、・・・
- ホームページおよびその作成ソフトウェア、・・・
- （インターネット）ブラウザ、・・・

いずれも、非常に長いプログラム

||

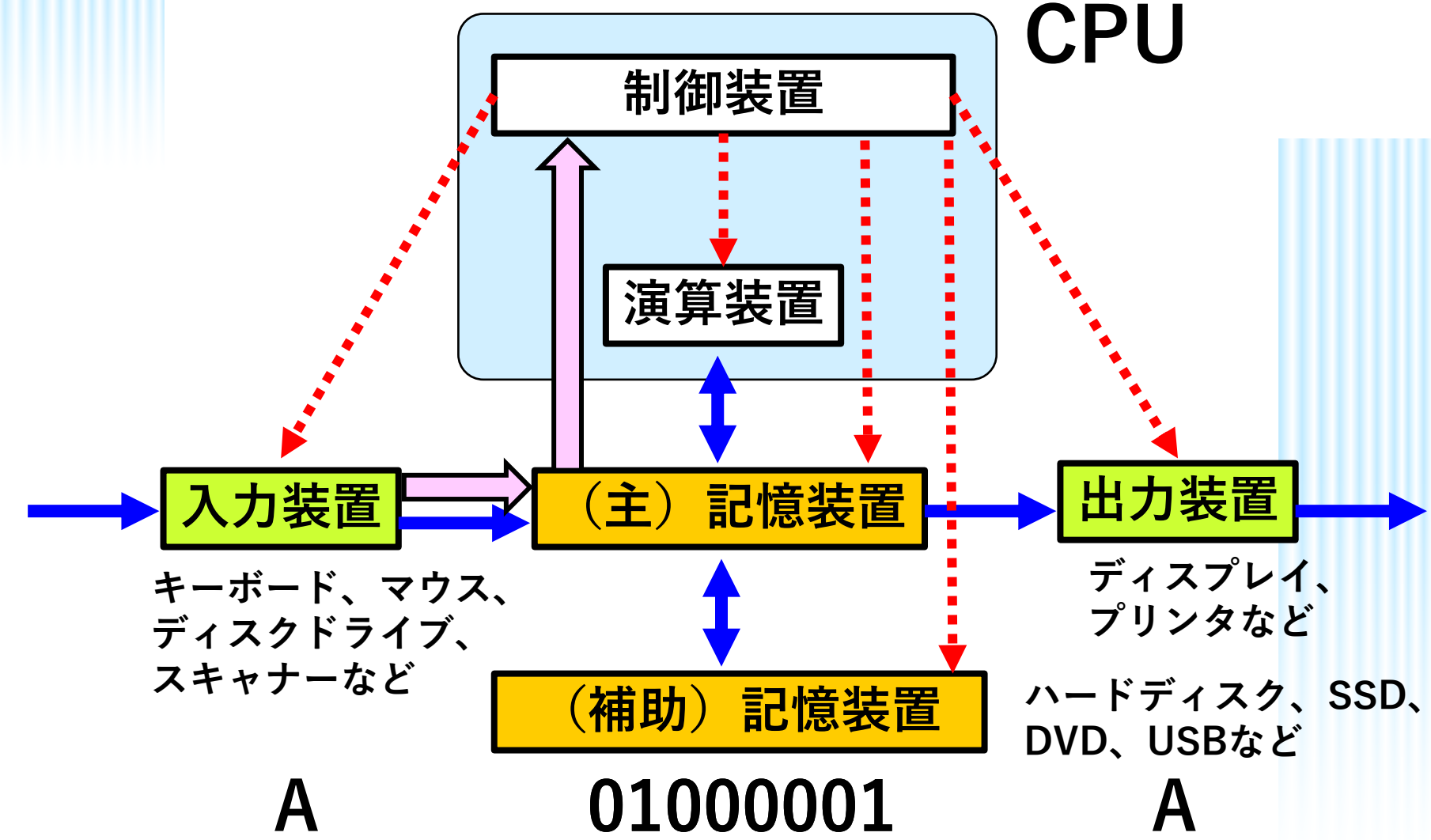
いわば、なが～い

お経 あるいは レシピ



1.4 コンピュータの素顔(3)

コンピュータの構成要素



1.4 コンピュータの素顔(4)

41へ

- 0と1だけの世界に生きる
- 繰り返 に強く、処理が 常に速い
- 数値 合との場 、誤 が含まれる
- プロ ム（作業 細な指示書）に従って動く あ に実施 自発的動作は
- プ 判断 の型など） 必要
- プログラムの き方は 非常に
- 書き方に少しでも違反が と動作せ 正要求をする
- 操作や内容についてチ 修正は ない 書き方が正しければその指示書通りに動く

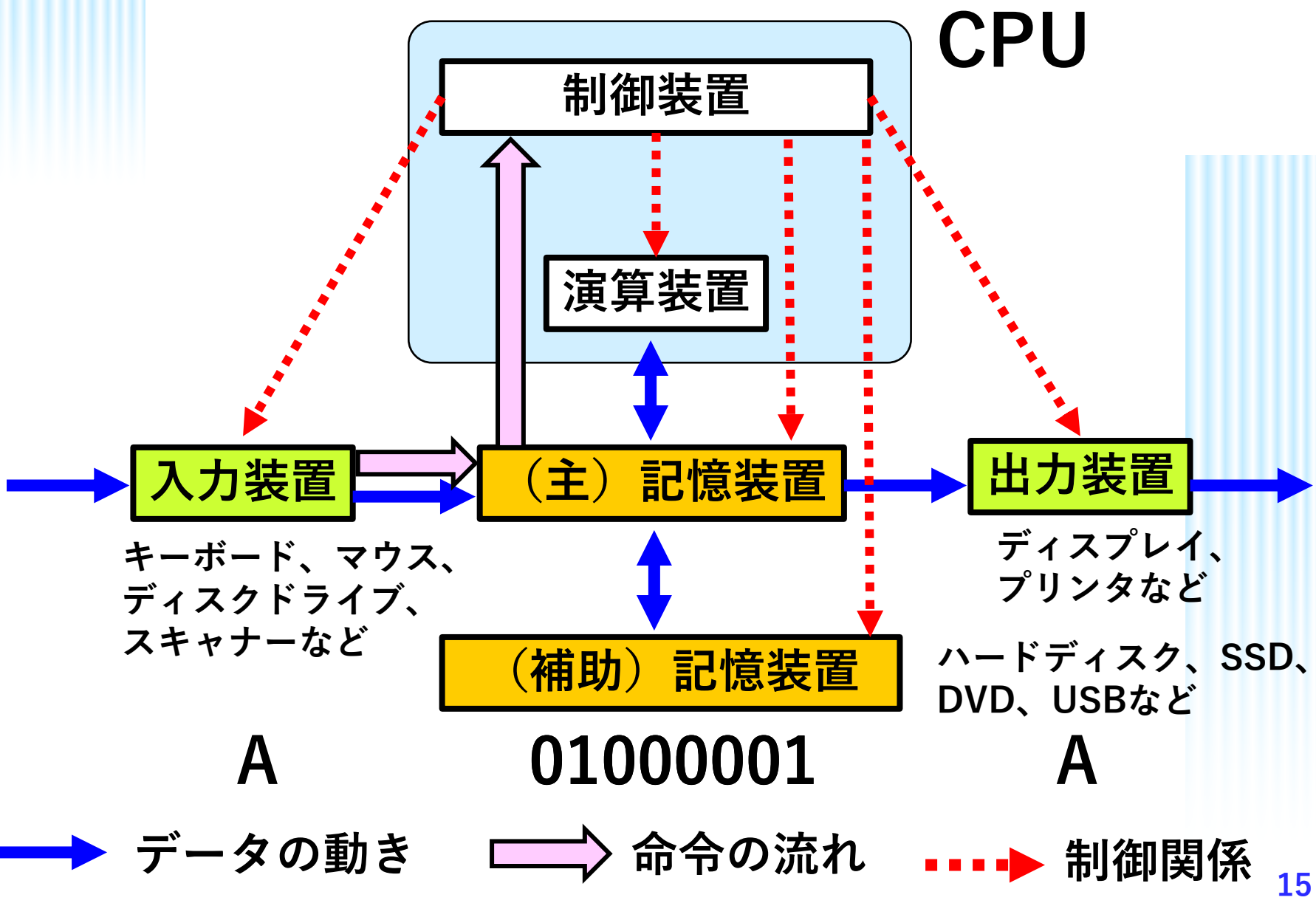
スライリー
TATSUMI MOOK
Carp SPIRITS 201
辰巳出版（株）, 20
り

1.4 コンピュータの素顔(5)

2. コンピュータの機能（ハードウェア）と情報処理の仕組み（ソフトウェア）

2.1 0と１だけの世界に生きる
2.2 数体系：数値の扱いの基本
2.3 コンピュータの仕組と内部データ
2.4 0.1を100個加えると？
2.5 コンピュータでのa+bの計算

2.1　0と1だけの世界に生きる



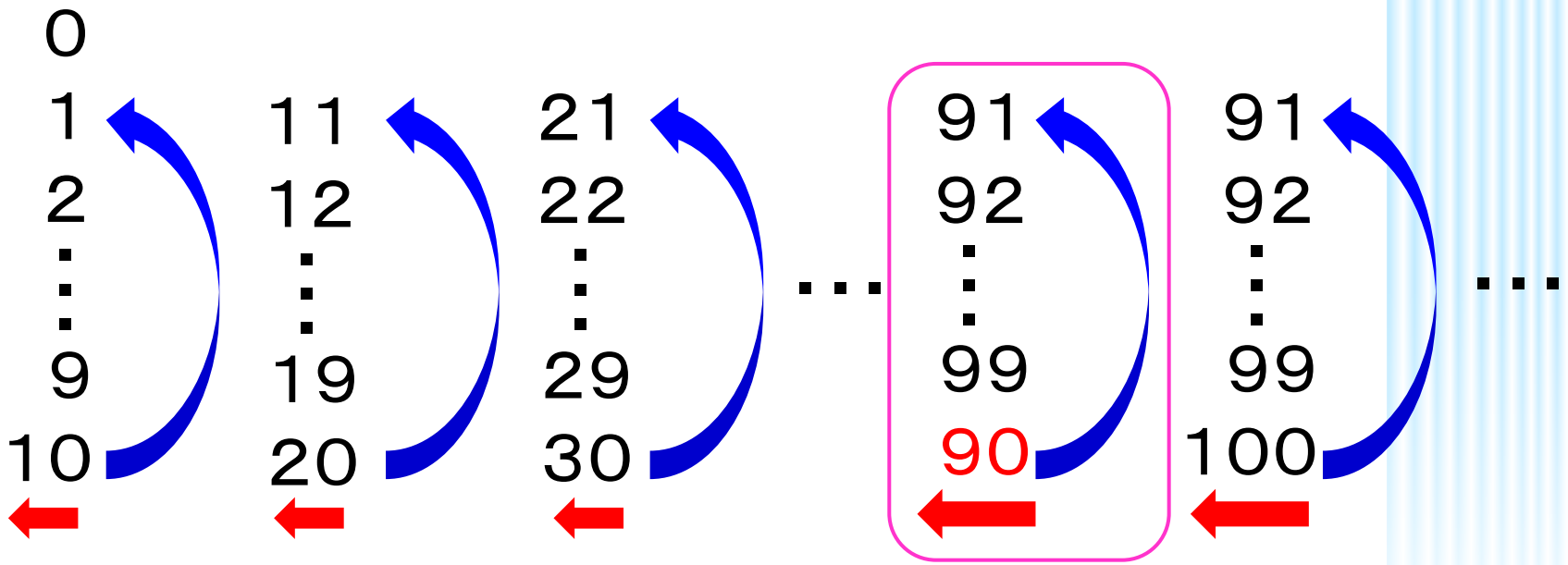
2.2 数体系：数値の扱いの基本

説明のポイント

- 数値（大きさを表す数字（文字））について
- 大きさの表現の仕組み（表記法）
- 10進法、8進法、2進法、16進法
- 表記法と数値の大きさとの関係
- 整数、小数点数、有理数の扱い

2.2 数体系(1)

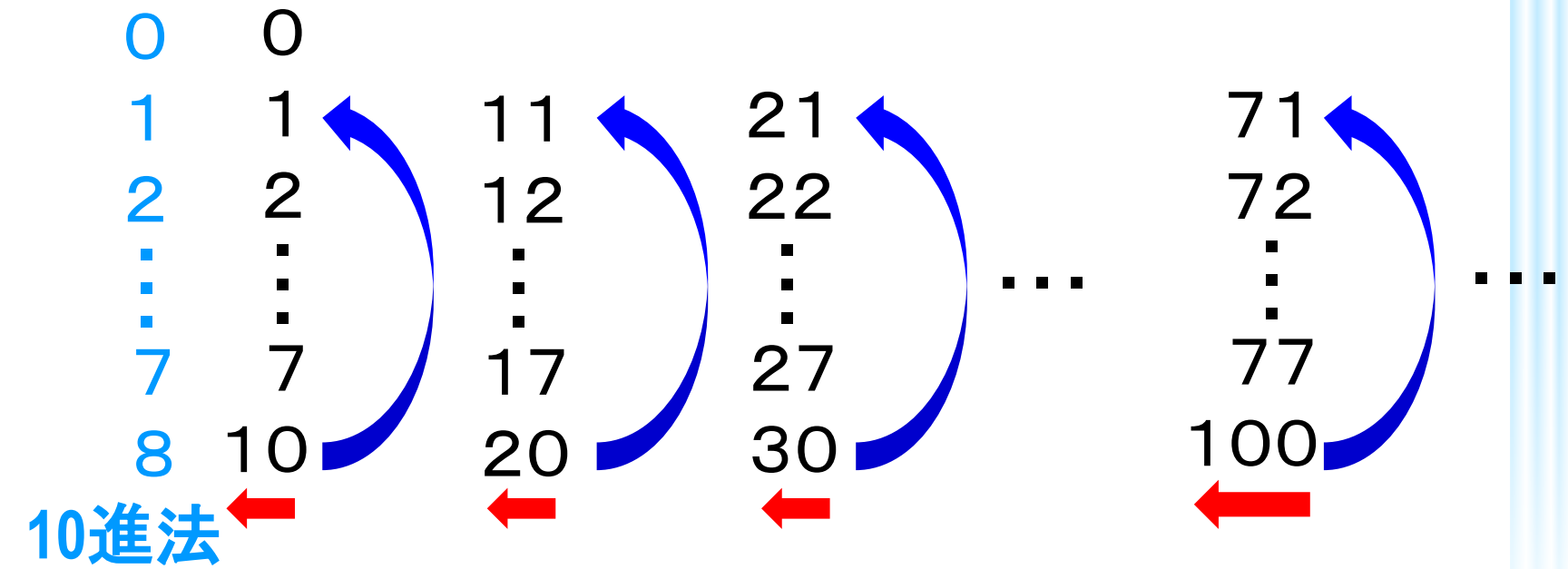
- 10進数の数値（基数10の数値表現）
- 使用するもの：０～９の10個の記号と桁



- （0→1→2→・・・→9 → 0と循環する
- 9 → 0のときには、左隣の桁の値を１増やす
- この２操作を左方向に反復

2.2 数体系(2)

- ８進数の数値（基数８の数値表現）
- 使用するもの：０～７の８個の記号と桁



- （0→1→2→・・・→7 → 0と循環する
- 7 → 0のときには、左隣の桁の値を１増やす
- この２操作を左方向に反復

2.2 数体系(3)

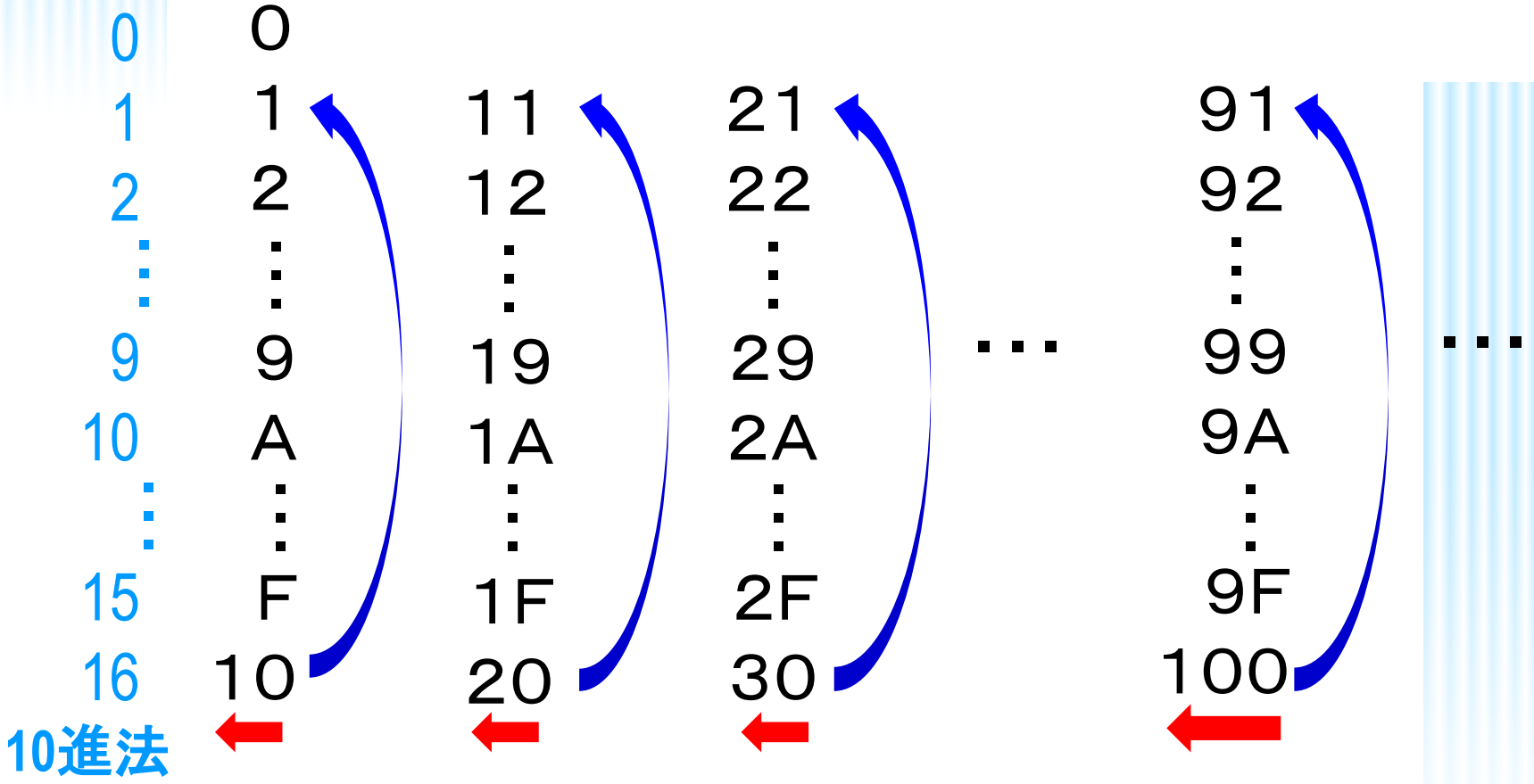
- （例） 2 進数の数値（基数 2 の数値表現）
- 使用するもの： 0、 1 の 2 個の記号と桁

10進法	2進法
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

- （0 → 1 → 0 と循環する
- 1 → 0 のときには、左隣の桁の値を 1 増やす
- この 2 操作を左方向に反復

2.2 数体系(4)

- １６進数の数値（基数１６の数値表現）
- 使用するもの：０～９、Ａ～Ｆの１６個の記号と桁



- （０→１→２→・・・→Ｆ → ０と循環する
- Ｆ → ０のときには、左隣の桁の値を１増やす
- この２操作を左方向に反復

2.2 数体系(5)
表記法

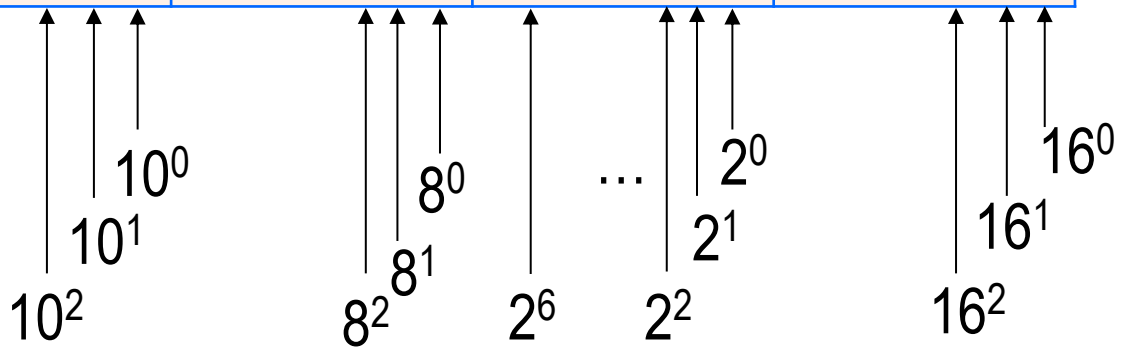
2.2.1 基数と桁

10進法	8進法	2進法	16進法	
0	0	0	0	
1	1	1	1	
2	2	10	2	
3	3	11	3	
4	4	100	4	
5	5	101	5	
6	6	110	6	
7	7	111	7	
8	10	1000	8	
9	11	1001	9	
10	12	1010	A	10
11	13	1011	B	11
12	14	1100	C	12
13	15	1101	D	13
14	16	1110	E	14
15	17	1111	F	15
16	**20**	**10000**	**10**	21

.....
19
20
21
.....
99
100

2.2.2 大きさの対応

10進法	8進法	2進法	16進法
.....	.....	.....	.....
16	20	10000	10
.....	.....	.....	.....
64	100	1000000	40



100	?	?	?

2.2.3 10進数 - 16進数（2進数）の表記

数値	表記	2進数
0	0x0	(0000)
1	0x1	(0001)
2	0x2	(0010)
3	0x3	(0011)
4	0x4	(0100)
5	0x5	(0101)
6	0x6	(0110)
7	0x7	(0111)
8	0x8	(1000)
9	0x9	(1001)
10	0xA	(1010)
11	0xB	(1011)
12	0xC	(1100)
13	0xD	(1101)
14	0xE	(1110)
15	0xF	(1111)

- 0x は16進数表記であることを表す

- 2進数4桁($2^4=16$)が16進数1桁に対応するので、機械的に置き換えることができる。

2.2.4 整数「10進→2進→8進,16進」なる変換

■ 10進数を 2で除算した余りを順に並べる。

$(100)_{10}$ → $(1100100)_2$

2) 100

2) 50 余り 0 下位（右側）

2) 25 余り 0

2) 12 余り 1

2) 6 余り 0

2) 3 余り 0

2) 1 余り 1

) 0 余り 1 上位（左側）

$(1100100)_2$

$(0110\ \ 0100)_2$

$(6)_{16}$ $(4)_{16}$

$(64)_{16}$

大きさ：16x6+1x4

$(001\ 100\ \ 100)_2$

$(1)_8$ $(4)_8$ $(4)_8$

$(144)_8$

大きさ：64x1+8x4+1x4

＊$(64)_{16}$ は 0x64 と表記することもある

2.2.5 正整数の「10進→2進→16進」なる変換

- 10進数を 2で除算した余りを順に並べる。

$(205)_{10}$ → $(11001101)_2$

2) 205
2) 102　余り　1
2) 51　余り　0
2) 25　余り　1
2) 12　余り　1
2) 6　余り　0
2) 3　余り　0
2) 1　余り　1
0　余り　1

$(11001101)_2$

2進数 $(1100\ \ 1101)_2$

0xC　0xD

16進数　0xCD

（注）負の整数は「２の補数表現」として扱うが、ここでは省略する

2.2.6 小数点数の「10進→２進」なる変換

- 10進数に２をかけて、できた整数部（０または１）を取り出し、順に並べる。

0.1875

x)　　2

0 ← 0.3750　上位

x)　2

0 ← 0.7500

x)　2

1 ← 1.5000

0.5000

x)　2

1 ← 1.0000　下位

0.0000　（ここで終了）

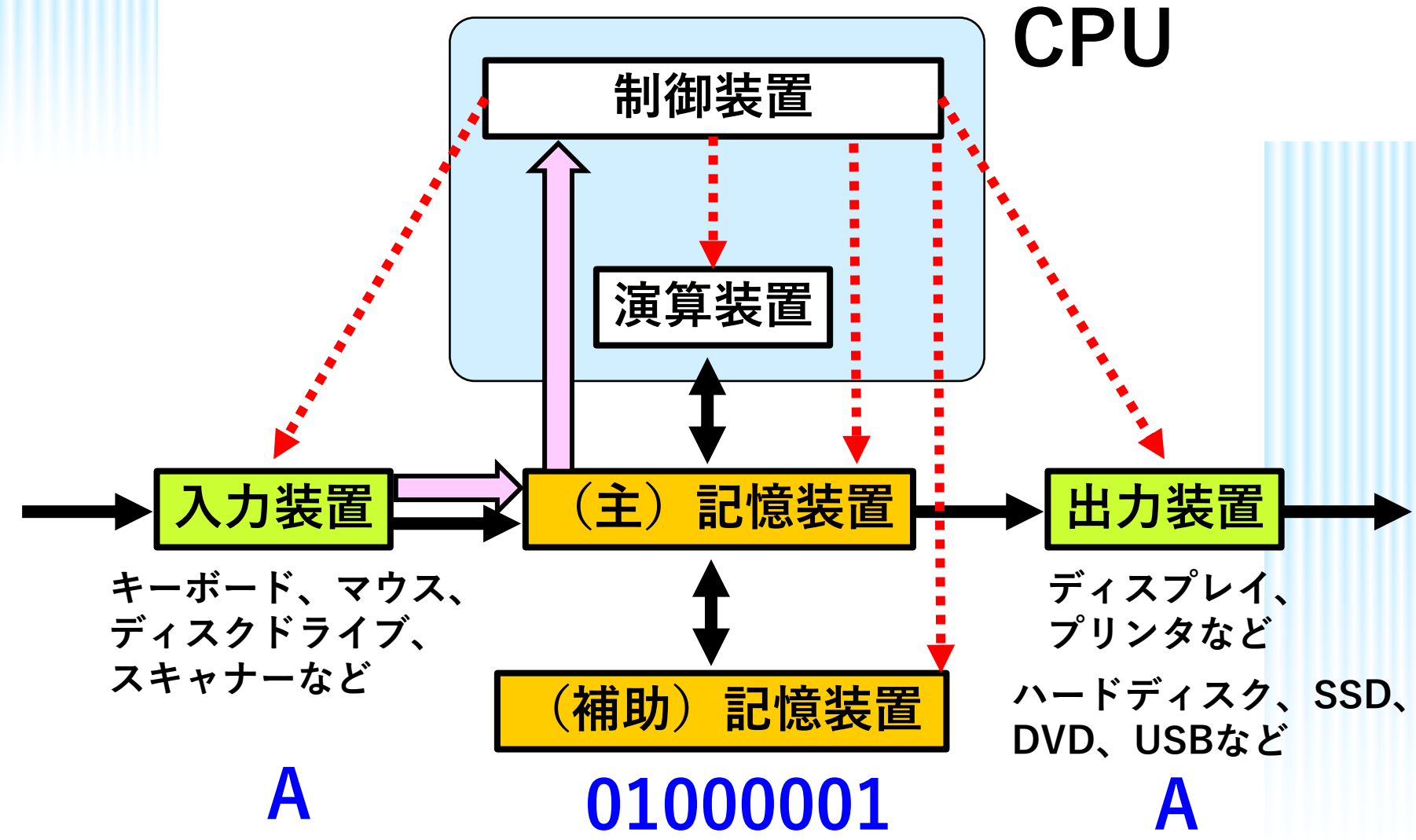
$(0.1875)_{10} = (0.0011)_2$

10^{-1} 10^{-2} 10^{-3} 10^{-4}

2^{-1} 2^{-2} 2^{-3} 2^{-4}

（注）負の有理数は補数記号（正は０；負は１）を付けて扱うが、ここでは省略する

2.3　コンピュータの仕組と内部データ



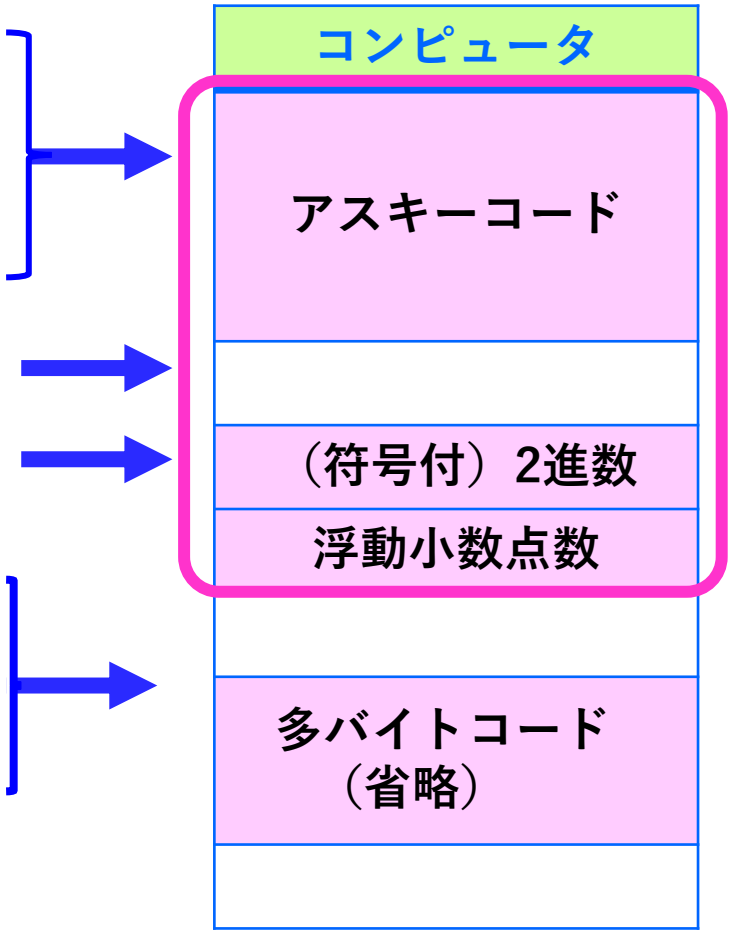
内部データとしての文字、数字（文字、数字の内部データ）

説明のポイント

- 文字（ひらがな、カタカナ、漢字、数字な）どがコンピュータ内部でどのようなデータとして扱われているか
- 文字コード：０、１のみを使ったコード（０、１の系列）
- アスキーコード（８ビット）
- 2バイトコード（16ビット）
- 符号化文字集合と文字化けについて

2.3.1 内部データ（符号化）

人間	
文字	A, …
数字	1, …
記号	=, …
・・・・・	
数値（整数）	13, …
数値（実数）	0.3, …
・・・・・	
かな	
漢字	
・・・・・	



0と1だけの系列

1ビット： 0または1
100万バイト＝1MB(メガバイト)
10億バイト＝1GB（ギガバイト）

1バイト： 8ビット（情報の基本）
1000GB ＝1TB（テラバイト）

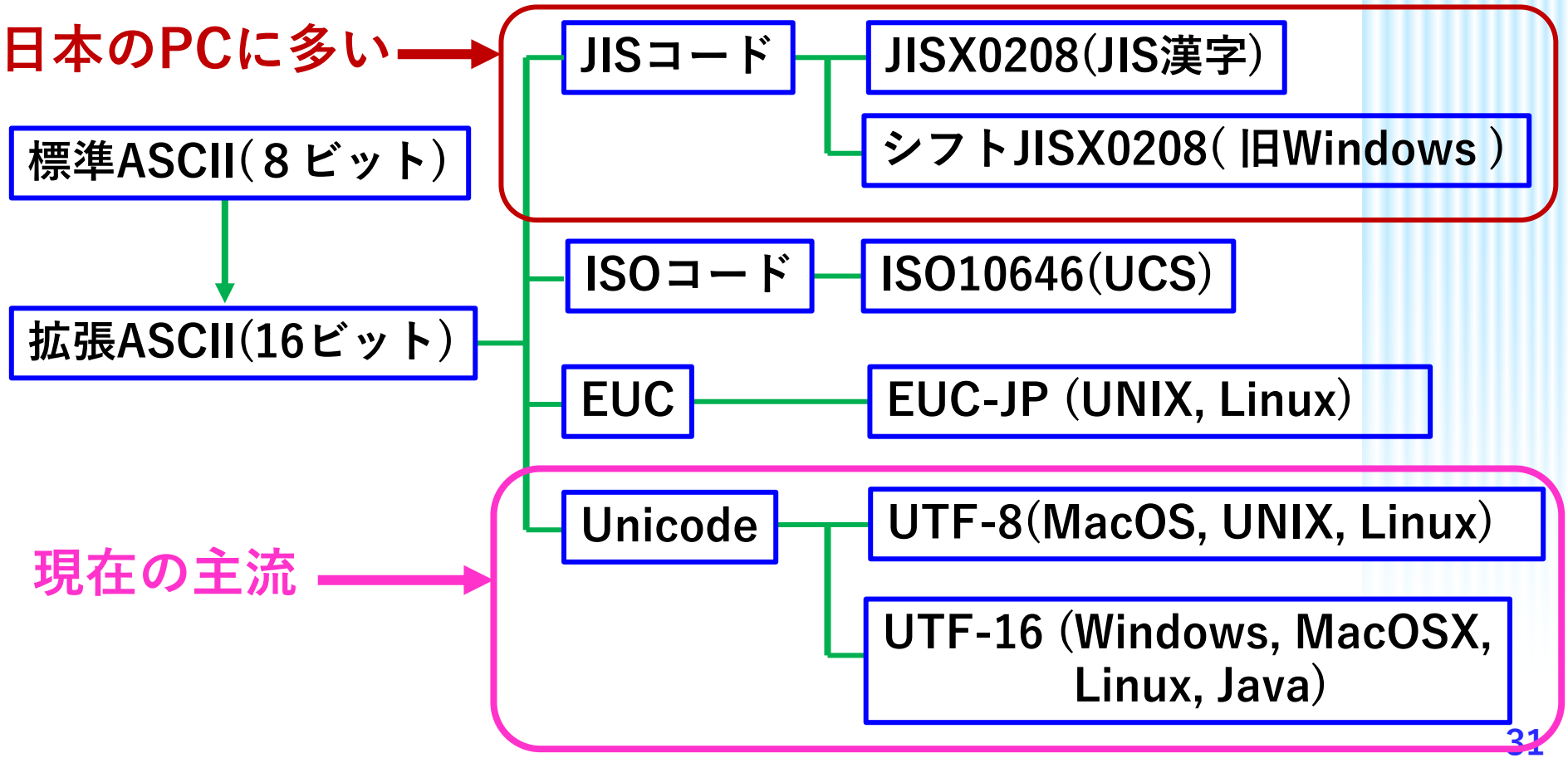
2.3.2 コンピュータ内部の文字データ

- A（半角）　0100 0001　8ビット＝1バイト
- a（半角）　0110 0001
- ＝（半角）　0011 1101
- ＝（半角）　0011 1101
- A（全角）　1111 1111 0010 0001 0000 0000
- そ（全角）　0011 0000 0101 1101 0000 0000
- 愛（全角）　1011 0000 1010 0110 0000 0000
- 13（整数）　0000 0000 0000 0000 0000 0000 0000 1101
- 13.0(実数)　0100 0001 0101 0000 0000 0000 0000 0000

注：以後、全角文字の区切り 0000 0000 は省略して表記することが多い

2.3.3 文字コードと符号化文字集合

- **文字コード**：文字や記号をコンピュータで表現するために割り当てた整数コード
- **符号化文字集合**：文字集合内の各文字を一意の非負整数に割り当てられたもの



2.3.4 （半角）文字、数字、記号、等
(1) アスキーコード

例	アスキーコード（８ビット）	１６進表記
A	0 1 0 0 0 0 0 1	41
x	0 1 1 1 1 0 0 0	78
1	0 0 1 1 0 0 0 1	31
7	0 0 1 1 0 1 1 1	37
=	0 0 1 1 1 1 0 1	3D
+	0 0 1 0 1 0 1 1	2B

(2) アスキーコード(ASCII Code)表

下位4 bit \ 上位3 bit		000	001	010	011	100	101	110	111
		0x 0	0x 1	0x 2	0x 3	0x 4	0x 5	0x 6	0x 7
0000	0x 0	NUL	DLE	SPC	0	@	P		p
0001	0x 1	SOH	DC1	!	1	A	Q	a	q
0010	0x 2	STX	DC2	"	2	B	R	b	r
0011	0x 3	ETX	DC3	#	3	C	S	c	s
0100	0x 4	EOT	DC4	$	4	D	T	d	t
0101	0x 5	ENQ	NAK	%	5	E	U	e	u
0110	0x 6	ACK	SYN	&	6	F	V	f	v
0111	0x 7	BEL	ETB	'	7	G	W	g	w
1000	0x 8	BS	CAN	(	8	H	X	h	x
1001	0x 9	HT	EM	)	9	I	Y	i	y
1010	0x A	LF	SUB	*	:	J	Z	j	z
1011	0x B	VT	ESC	+	;	K	[	k	{
1100	0x C	FF	FS	,	<	L	¥	l	|
1101	0x D	CR	GS	-	=	M	]	m	}
1110	0x E	SO	RS	.	>	N	^	n	~
1111	0x F	SI	US	/	?	O	_	o	DEL

(3) 制御文字

コード	記号	説明
0x00	NUL	空文字
0x01	SOH	ヘディング開始
0x02	STX	テキスト開始
0x03	ETX	テキスト終了
0x04	EOT	伝送終了
0x05	ENQ	問い合わせ
0x06	ACK	肯定応答
0x07	BEL	ベル
0x08	BS	後退
0x09	HT	水平タブ
0x0A	LF	改行
0x0B	VT	垂直タブ
0x0C	FF	改頁
0x0D	CR	復帰
0x0E	SO	シフトアウト
0x0F	SI	シフトイン

コード	記号	説明
0x10	DLE	データリンク拡張
0x11	DC1	装置制御 1
0x12	DC2	装置制御 2
0x13	DC3	装置制御 3
0x14	DC4	装置制御 4
0x15	NAC	否定応答
0x16	SYN	同期信号
0x17	ETB	伝送ブロック終了
0x18	CAN	取消
0x19	EM	媒体終端
0x1A	SUB	置換キャラクタ
0x1B	ESC	拡張
0x1C	GS	グループセパレータ
0x1D	FS	ファイル分離
0x1E	RS	レコード分離
0x1F	US	ユニット分離
0x7F	DEL	削除

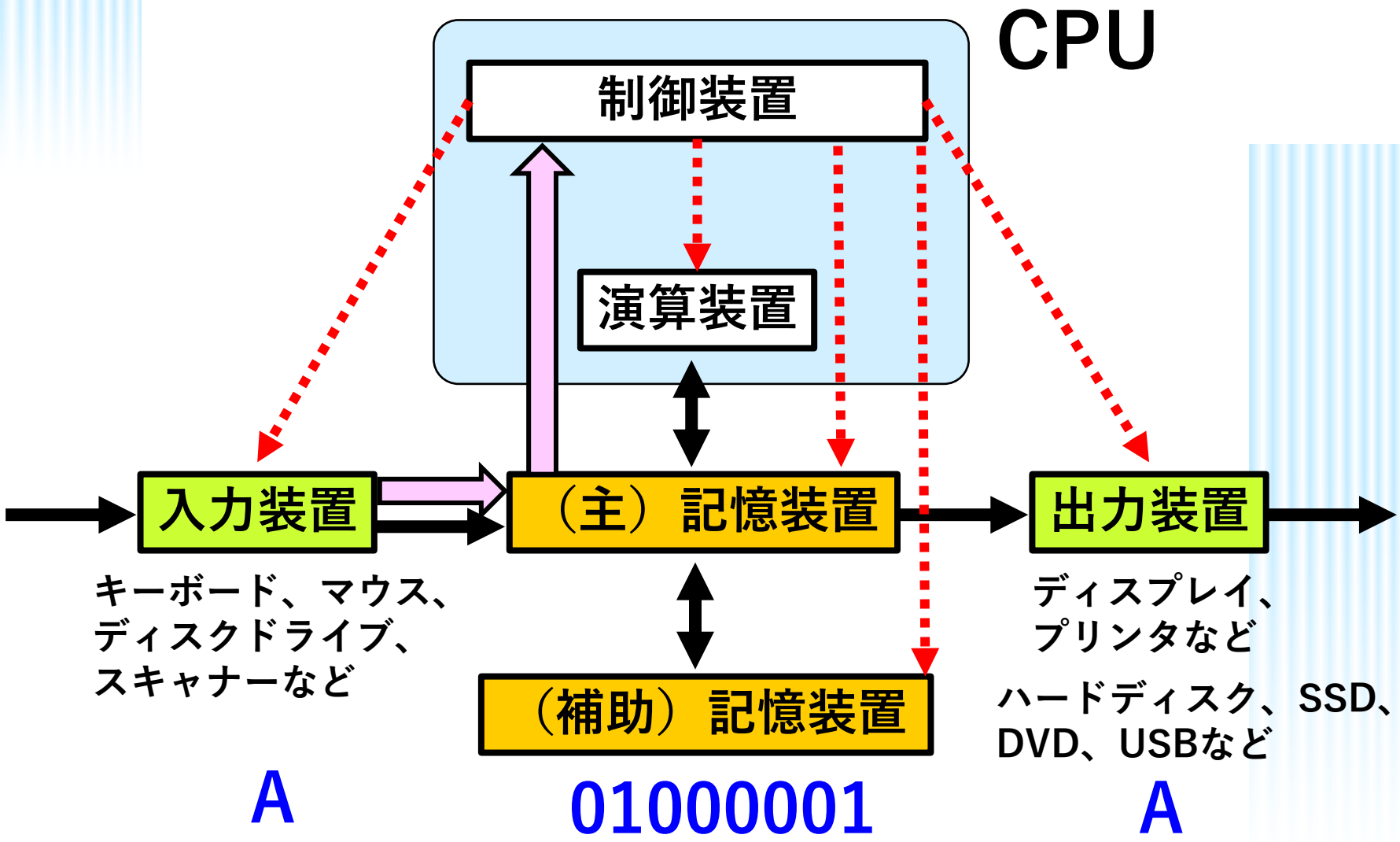
(4) アスキーコード表の使い方

- 例えば、'A'のアスキーコードは表の『A』の文字がある場所を見つけ、
  上位3bit　　0x4(16進数)
  下位3bit　　0x1(16進数)
- まとめると、0x41(16進数)となる。
- 10進数に直すと、4×16＋1＝65 となる。
- アスキーコードは1byte文字である。
- 1byte=8bit（0または1が８個並んだ系列）

(5) JISコード表

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø				Ø	@	P	‘	p				ー	タ	ミ		
1			!	1	A	Q	a	q			。	ア	チ	ム		
2			"	2	B	R	b	r			「	イ	ツ	メ		
3			#	3	C	S	c	s			」	ウ	テ	モ		
4			$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	e	u			・	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	\a		’	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	\b		(	8	H	X	h	x			ィ	ク	ネ	リ		
9	\t		)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	\n		*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	\v		+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C	\f		,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	\r		−	=	M	]	m	}			ュ	ス	ヘ	ン		
E			.	>	N	^	n	¯			ョ	セ	ホ	゛		
F			/	?	O	_	o				ッ	ソ	マ	゜		

コンピュータの仕組み（再掲）



(6) 愛 は －ヲ？

- 文字符号化方式違いによる文字化けの一例

愛（全角）1011 0000 1010 0110　日本語EUC

―　ヲ　ShiftJIS（JIS漢字）

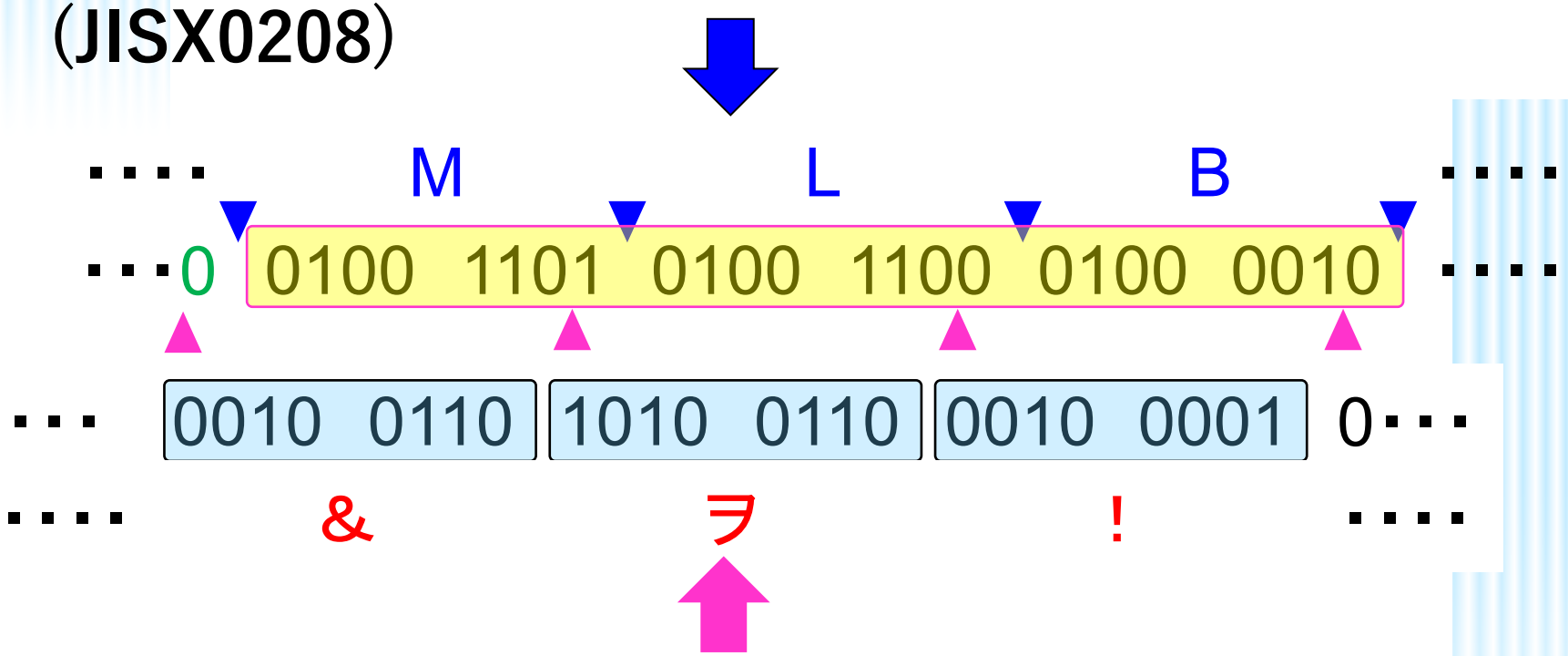
Aさん
文章作成、送付
（日本語EUC）
愛

渡す

Bさん
文章を読む
（ShiftJIS（JIS漢字））
－ヲ

(7) MLB のお隣さんは ＆ヲ！

- ・・・MLB・・・とキーボードを押すと内部では(JISX0208)



- Mの左側の文字の0,1系列の最後が0とし、コンピュータが間違ってこの0から（１ビット左から）文字を変換したとすると、・・・＆ヲ！・・・となる （文字化けの原理）

2.3.5 コンピュータでの数値の扱いと誤差

- 正の整数について（負の場合は省略）
- 正の小数点数について（負の場合は省略）
- 実数（有理数）の扱いー丸め誤差ー
- 計算における誤差の集積と誤差補正について

(1) 整数は？

- 整数：13, 2, 15, …

  コンピュータ内部では2進数で扱う（誤差なし）

- 例：13+2=15 の具体的な姿（32ビットの場合）

13:　0000 0000 0000 0000 0000 0000 0000 1101

2:　0000 0000 0000 0000 0000 0000 0000 0010

15:　0000 0000 0000 0000 0000 0000 0000 1111

0：正　1：負　（厳密には補数表示を意味するが、ここでは省略）

扱える範囲（大まかに）：－20億～20億

（－2,147,483,648 ～ 2,147,483,647）

(2) 実数は？

・実際には、有理数（小数点数）として扱う*

・数値と数字（文字として）の違い（例）

13（整数）　0000 0000 0000 0000 0000 0000 0000 1101

13.0（実数）0100 0001 0101 0000 0000 0000 0000 0000

13（数字）　0011 0001 0011 0011 0000 0000 0000 0000

数値：計算に使う

文字：計算には使えない

$13.0=(1101.0)_2=(1.101)_2 \times 2^3$ → 指数：3+127=130=$(10000010)_2$

仮数：(1.の部分は省略して)$(101)_2$

整数とみると $2^{30}+2^{24}+2^{22}+2^{20} \fallingdotseq 1.0957\text{X}10^9$ (約 10億9千6百万)

＊IEEE754浮動小数点規格

(3) 13.0 と 13 は同じ？ －データ型の指定－

- 実数：13.0, 0.2, $\sqrt{2}$, π=3.1415…, …

  浮動小数点数として扱う

- 例：13.0+2 は 15.0 または15のどちら?

  具体的な姿（32ビットの場合）

13: 0000 0000 0000 0000 0000 0000 0000 1101

13.0: 0100 0001 0101 0000 0000 0000 0000 0000

2: 0000 0000 0000 0000 0000 0000 0000 0010

13.0+2:0100 0001 0101 0000 0000 0000 0000 0010

整数とすると $2^{30}+2^{24}+2^{22}+2^{20}+2 > 15$

浮動小数点数とすると $13.0+2^{-22} < 15$

(4) 13.0+2の値は?

- 13+2=15 → 1101+0010=1111
- 13.0+2.0=15.0 → 1101.0+0010.0 → $0.1101 \times 2^4 + 0.1 \times 2^2$
- 13.0+2= → 13.0+2.0=15.0
- 整数と浮動小数点数の混在は、すべて浮動小数点数として扱う

13: 0000 0000 0000 0000 0000 0000 0000 1101

2: 0000 0000 0000 0000 0000 0000 0000 0010

15: 0000 0000 0000 0000 0000 0000 0000 1111

13.0: 0100 0001 0101 0000 0000 0000 0000 0000

2.0: 0100 0001 0010 0000 0000 0000 0000 0000

15.0: 0100 0001 0111 0000 0000 0000 0000 0000

＊ $(1.101)_2 \times 2^3 + (0.010)_2 \times 2^3 = (1.111)_2 \times 2^3 = (1111.0)_2 = (15.0)_{10}$

2.4 0.1を100個加えると?

説明内容

2.4.1　0.1を100個加えると

- 0.1を100個加えると、10になる
- 0.3を100個加えると、30になる
- 0.5を100個加えると、50になる
- 実際にコンピュータで計算してみると

x	n	s
0.1	100	10.000002
0.2	100	20.000004
0.3	100	29.999971
0.4	100	40.000008
0.5	100	50.000000

2.4.2 0.1の内部での姿（32ビット版）

-4+127=123

3DCCCCCD$_{(16)}$

0.1: 0011 1101 1100 1100 1100 1100 1100 1101

10進数では：0.100000014901161193847 7

1100

$(0.1)_{10}= (0.000110011001100\ldots)_2= (1.10011001100\ldots)_2 \times 2^{-4}$

-2+127=125

3E99999A$_{(16)}$

0.3: 0011 1110 1001 1001 1001 1001 1001 1010

10進数では：0.300000011920928955078 12

1001

$(0.3)_{10}= (0.0100110011001\ldots)_2= (1.00110011001\ldots)_2 \times 2^{-2}$

-1+127=126

3F000000$_{(16)}$

0.5: 0011 1111 0000 0000 0000 0000 0000 0000

10進数では：0.500000000000000000000

0000

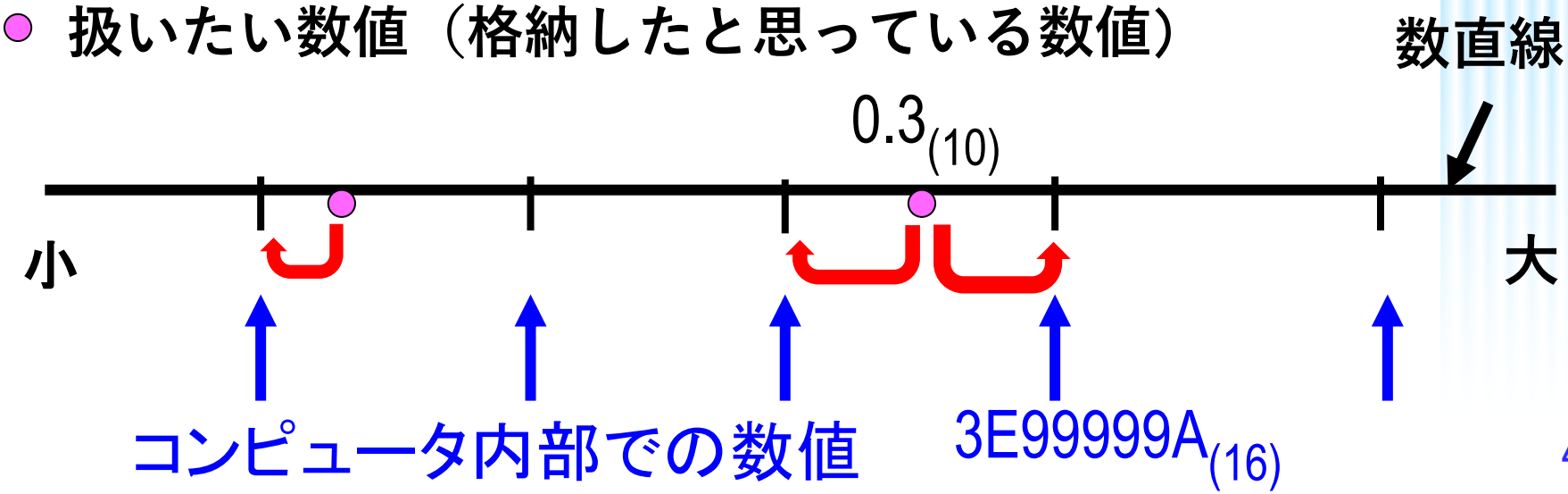
$(0.5)_{10}= (0.1)_2= (1.0)_2 \times 2^{-1}$ **誤差なし（真値）**

2^{-23} 以下の誤差

2.4.3 数値は誤差がいっぱい

- 浮動小数点数の有効
  - 単精度で6～7桁（10^{38}）
  - 倍精度（64ビット）で15桁（10^{-307} ～ 10^{308}）
- 0.0が無い：絶対値最小は 10^{-38}＞0.0
  - A-B=0.0 でも A=B は違う可能性あり

- **丸めによる誤差が頻繁に発生：近い数値に置き換える**
  - 扱いたい数値（格納したと思っている数値）



2.4.4 0.1, 0.3, 0.5を各々100個加えた値

x	n	s
0.1	100	10.000002
		10.0000019073486328125000
0.3	100	29.999971
		29.99997138977050781250000
0.5	100	50.000000 ← 誤差なし（真値）
		50.00000000000000000000

2.4.5 0.3を100個加える過程を見ると

x	n	s	
0.3	61	18.300001 > 18.3	
0.3	62	18.600000* > 18.6	*誤差なし(見かけ)
0.3	63	18.900000** > 18.9	**誤差なし(見かけ)
0.3	64	19.199999 < 19.2	
0.3	65	19.499998 < 19.5	

もっと精度をあげてみると

0.3	62	18.600000000000002273736754
0.3	63	18.900000000000002344791028

2.4.6 数値計算は誤差に注意を(1)

- 整数に誤差はなし。ただし、大きさの範囲に注意：外れると通常は停止する
- **小数点数（浮動小数点数）は誤差を含んで扱われる：扱える近い数値に（黙って）置き換えられる**
- 大きさの近い数値の引き算は極力回避しよう
- A=Bの判定にA-B=0.0を使わない
- 繰り返し回数の判定は整数で（小数点数は使わない）

2.4.6 数値計算は誤差に注意を(2)

- 誤差が入りそうな場合その都度補正（誤差除去）を
  - ２数の引き算をした場合２乗して得た項を多数加える、さらに割り算（分散や標準偏差の計算で出現）
- 時には「数値計算法」を調べて誤差の少ない計算法を考えてみよう
- **エクセルがいつも正しいとは限らない：エクセルがカバーできない（補正できない）場合もある**

講義前半のまとめ

- コンピュータは魔法使い？
- コンピュータは台所とお母さん
  - コンピュータの仕組をたとえ話で説明
- 現代社会におけるコンピュータの役割（概要）
- コンピュータの素顔
  - 数体系：数値の扱いの基本
  - コンピュータの仕組みと内部データ
  - 数値計算に関する注意
- コンピュータの機能（ハードウェア）と情報処理の仕組み（ソフトウェア）

3. 情報の収納方法(データ構造)や処理手順(アルゴリズム)

3.1 アルゴリズムとは

3.2 コンピュータを直接に操る

3.3 アルゴリズムデザイン

3.4 アルゴリズムの評価について

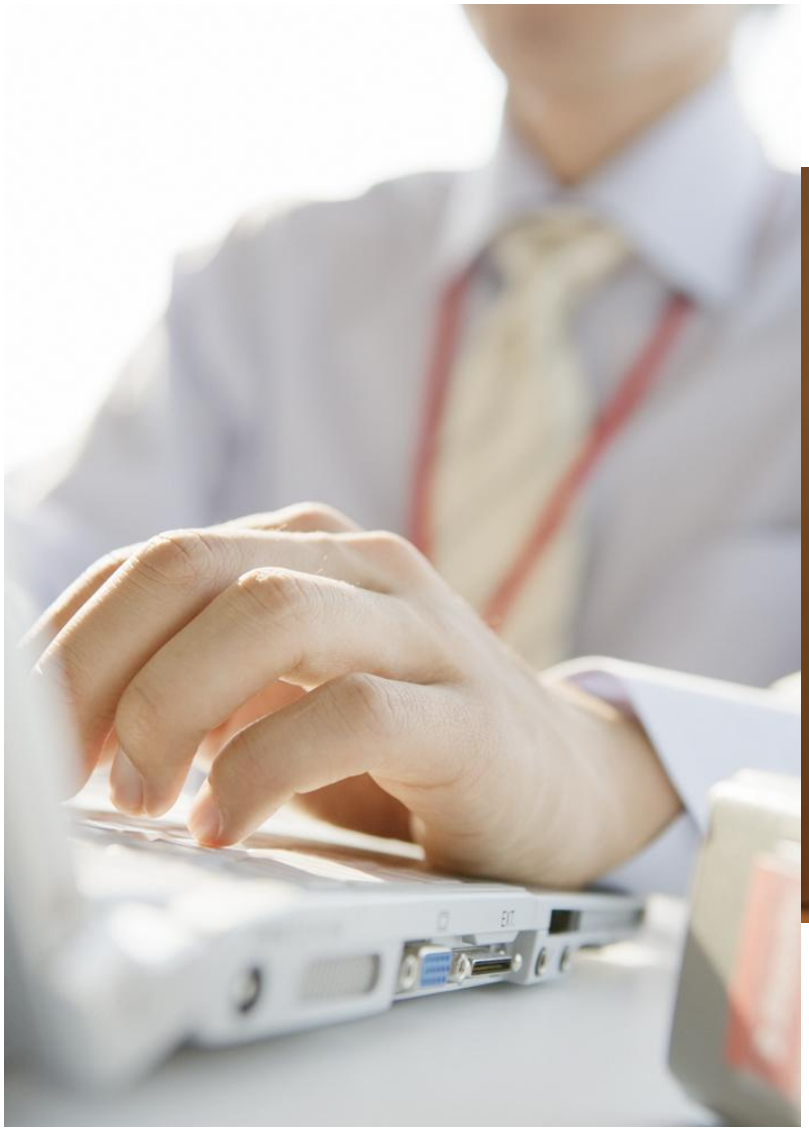
3.5 コンピュータでのa+bの計算
(プログラミングと内部処理の具体例)

3.1 アルゴリズムとは

- アルゴリズム：処理や作業の手順
- データ構造：メモリ内でのデータの蓄え
  リズムを効率よく実行するため）
- アルゴリズムデザインと目標：
  - 処理時間が短く、使用メモリ量が少ない
  - 結果が正確で（誤差が少なく）、誤動作が無い（発生しにくい）

  アルゴリズムの設計（データ構造も含む）
- プログラム：アルゴリズムとデータ構造をプログラミング言語（書き方のルール）で詳細に記述した（コンピュータへの）指示書（台所とお母さんの「レシピ」に対応）

3.2 コンピュータを直接に操る



3.3 アルゴリズムデザイン

説明のポイント

- 多項式計算を例とした説明
- 初等的計算法とホーナー法の紹介
- 演算回数、使用メモリー量の比較
- アルゴリズム設計と評価
  ー計算時間と使用メモリ量ー
- ホーナー法の具体的な実行状況

3.3.1 アルゴリズム設計の実際
－式の計算を例として－

［例題１］ 以下の式のa=2のときの値計算

$$f(a)=4a^3+5a^2+3a+7$$

- 計算で使用するデータ（コンピュータのメモリに格納しておく初期データ：いつでも使用可）

最大次数	a^3の係数	a^2の係数	aの係数	定数項	aの値
3	4	5	3	7	2

- 設計目標：演算回数と記憶する途中結果を少なくする
  - 誤差の混入防止； 処理高速化； 使用メモリ量の減少
- 説明の都合で整数だけの計算を実行する

3.3.2 方法１（初等的計算法）

- 初期データ（共通）　$f(a)=4a^3+5a^2+3a+7$

最大次数	a^3の係数	a^2の係数	aの係数	定数項	aの値
3	4	5	3	7	2

- 第１項＝4×2×2×2＝32←記憶する途中結果　3回
- 第２項＝5×2×2＝20←記憶する途中結果　2回
- 第３項＝3×2＝6←記憶する途中結果　1回
- 式の値＝32＋20＋6＋7＝65←記憶する答　3回
- 掛け算：6回　　演算回数：9回
- 記憶する途中結果：3個（別々に記憶）
- 答：１個（途中結果とは別に記憶しておくことにする）

皆さんへのクイズです
これらの数値を使って、式の値を求めてみてください
そのときに、どのような計算をしたか
どのような途中結果を記憶していたか
など覚えておいてください

3.3.3 方法２（ホーナー法）

- 初期データ（共通）　$f(a)=4a^3+5a^2+3a+7$

最大次数	a^3の係数	a^2の係数	aの係数	定数項
3	4	5	3	7

- １回目（ 4a+5 の計算 ）　xが
  4x2+5= 13 ←記憶する途中結果
- ２回目（ （4a+5）a+3 の計算 ）　xが１回　+が1回
  13x2+3= 29 ←記憶する途中結果(13に上書き)
- ３回目（ （（4a+5）a+3）a+7　xが１回　+が1回
  29x2+7= 65 ←記憶する途中結果(29に上書き)
- 掛け算：３回　　足し算：３回　　演算回数：６回
- 記憶する途中結果（答も合わせて常に）：１個

3.3.4 方法１と方法２の比較

アルゴリズム
として採用

	方法１				方法２			
最大次数	3	10	20	n	3	10	20	n
乗算回数	6	55	210	n(n+1)/2	3	10	20	n
演算回数	9	65	230	n(n+3)/2	6			n
途中結果	3	10	20	n	1			1
答	1	1	1	1				

（注）　加算回数は最大次数に等しい

26へ

C言語で書いたプログラム
まるでお経ですね
眺めるだけにしましょう

```
/* Program name: h_ex.c */
/* Horner's Method */
#include <stdio.h>
main()
{
  int n, i, a, k;
  int b[1000];
  printf("¥n");
  printf("多項式の計算(Horner's Rule)¥n");
  printf("¥n");
  printf("最大次数を入力して下さい: ");
  scanf("%d",&n); printf("¥n");
  printf("各項の係数を入力して下さい¥n");
  for(i = n; i >= 0; i--){
    printf(" 次数が%2d の項の係数: ",i);
    scanf("%d",&b[i]);
  }
  printf("¥n");
```

25へ

お経の続きです

```
  printf("a の値を入力して下さい: ");
  scanf("%d",&a); printf("¥n");
  printf("----------------------------¥n");
  printf("入力した多項式 (ただし、a^rはaのr乗) ¥n");
  printf("¥n");
  for(i = n; i >= 1; i--)
    printf(" %d*%d^%d +",b[i],a,i);
  printf(" %d*%d^%d ¥n",b[i],a,i);
```

```
  k = b[n];
  for(i = n-1; i >= 0; i--)
    k = k*a+b[i];
```

この部分が
ホーナー法ですが
眺めるだけでOK

```
  printf("¥n 答:%d¥n",k); printf("¥n");
}
```

3.3.5 例題１の計算

［例題１］ 以下の式のa=2のときの値を計算する

$$f(a)=4a^3+5a^2+3a+7$$

- 計算で使用する初期データ（コンピュータのメモリに格納する：キーボードから入れる）

n	b[3]	b[2]	b[1]	b[0]	a
3	4	5	3	7	2

走らせてみると

: : 多項式の計算（Hornr’s Rule）: :

最大次数を入力して下さい: ■ 3

各項の係数を入力して下さい
次数が 3 の項の係数: ■ 4
次数が 2 の項の係数: ■ 5
次数が 1 の項の係数: ■ 3
次数が 0 の項の係数: ■ 7

走らせてみると(続き)

aの値を入力して下さい: ■ 2

入力した多項式 （但し、a^rはaのr乗を表す）

4＊2^3 + 5＊2^2 + 3＊2^1 + 7＊2^0

答:65

33へ

3.3.6 別の例題の計算
－プログラムの再実行－

［例題２］ 以下の式のa=5のときの値を計算

$$g(a)=4a^4+5a^3+3a^2+7a+6$$

- 計算で使用する初期データ（コンピュータのメモリに格納する：キーボードから入れる）

n	b[4]	b[3]	b[2]	b[1]	b[0]	a
4	4	5	3	7	6	5

再び走らせてみると

: : 多項式の計算（Hornr's Rule）: :

最大次数を入力して下さい: ■ 4

各項の係数を入力して下さい
次数が 4 の項の係数: ■ 4
次数が 3 の項の係数: ■ 5
次数が 2 の項の係数: ■ 3
次数が 1 の項の係数: ■ 7
次数が 0 の項の係数: ■ 6

再び走らせてみると(続き)

aの値を入力して下さい: ■ 5

入力した多項式 （但し、a^rはaのr乗を表す）

4＊5^4 + 5＊5^3 + 3＊5^2 + 7＊5^1 + 6＊5^0

答:3241

3.3.7 例題１の計算

［例題１］ 以下の式のa=2のときの値を計算

$$f(a)=4a^3+5a^2+3a+7$$

- 計算で使用する初期データ（コンピュータのメモリに格納する：キーボードから入れる）

n	b[3]	b[2]	b[1]	b[0]	a
3	4	5	3	7	2

33へ

3.3.8.別の例題の計算
－プログラムの再実行－

［例題２］ 以下の式のa＝5のときの値を計算

$$g(a)=4a^4+5a^3+3a^2+7a+6$$

- 計算で使用する初期データ（コンピュータのメモリに格納する：キーボードから入れる）

n	b[4]	b[3]	b[2]	b[1]	b[0]	a
4	4	5	3	7	6	5

3.3.9 例題１、例題2の共通性

n, a および係数 $k_n, k_{n-1}, \cdots, k_1, k_0$ を具体的に与えて（キーボードから打ち込んでコンピュータのメモリに格納し）、以下の式の値 f(a) を計算する：

$$f(a) = k_n a^n + k_{n-1} a^{n-1} + \cdots + k_1 a + k_0$$

このような共通性をプログラミング言語で記述することで(コンピュータ)プログラムができる。

3.4 アルゴリズムの評価について

説明のポイント

- アルゴリズムの評価基準
- 最悪計算時間と使用メモリ量の具体例
- 二分木探索を例とした比較

3.4.1 アルゴリズムの評価基準

- 計算時間
  - 最悪計算時間←これを採用することが一般的
  - 平均計算時間←算出が困難な場合が多い
- 使用メモリ量
- これらの比較によって、アルゴリズムの優劣を決定する。
- 正当性（正しく処理が実行されること）は前提とする。
- アルゴリズム設計に際しては、正当性、計算時間、使用メモリ量のチェックが必須

30へ

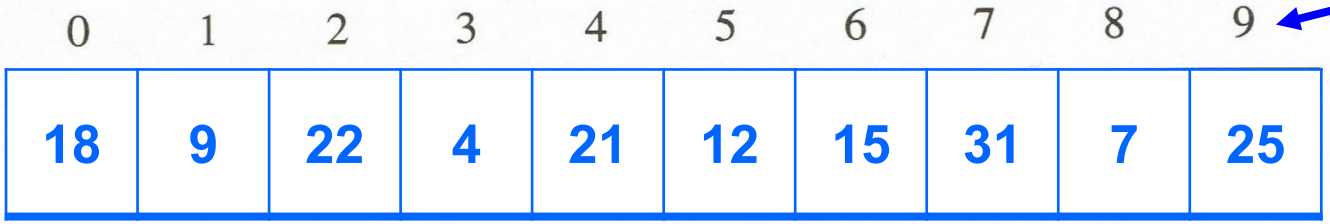
3.4.2 いろいろな基本的アルゴリズム

- **整列:**多数の数値（たとえば整数）を小から大へ（または大から小へ）順に**並べ替える**
  - 例１：全受験者を受験番号順に並べ替える
  - 例２：成績順に並べ替（高得点から順に、など）
- **探索:**多数のデータの中から、ある条件を満たすものを**探す**（該当するものの有無判定も含む）(例)受験者の中からある条件をみたす人をすべて選ぶ
- **最短パス探索**
  - 指定した２点A、Bについて、AからBへ最短（距離総和が最小）の**道を見つける**
  - **処理時間が短く**、使用メモリ量が少ない

3.4.3 探索アルゴリズムによる説明

１次元配列

（メモリ内の）識別番号

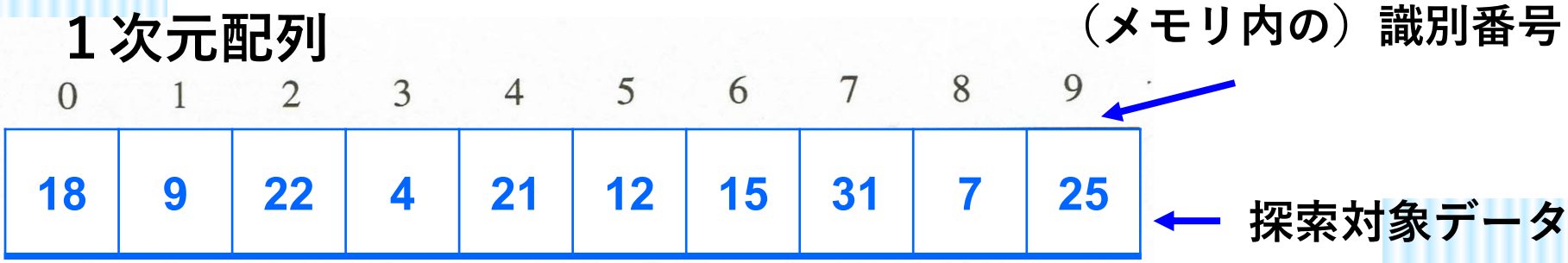


探索アルゴリズム member(x):　与えられた集合Sの中に要素xがあるか否か調べ、あればYesを、なければNoを出力する。

（大まかな処理手順）

Step 1：未探索データにアクセスする。
Step 2：データを取り出してxと比較する。
Step 3：取り出したデータがxに等しければYesを出力して終了する。
　　等しくないときに、
　　もし全データを探索していればNoを出力して終了する。
　　もしまだ未探索データが残っていれば、Step 1 に戻る。

3.4.4 最悪計算時間



Step 1～Step 3をひとまとまりの処理と考え、
これがC単位時間で実行できるとする。終了までに実行した回数を調べてみる。実行時間は（C x 回数）単位時間。

初等的探索：１次元配列を左から右へ（あるいはその逆に）見てゆく

	左から	右から
member(4)	4	7
member(15)	7	4
member(16)	10	10
member(22)	3	7
member(7)	9	2

最悪計算時間は10C
要素数がnならばnC
より速いアルゴリズムは？

2.3 (平衡)二分探索木の利用(1)

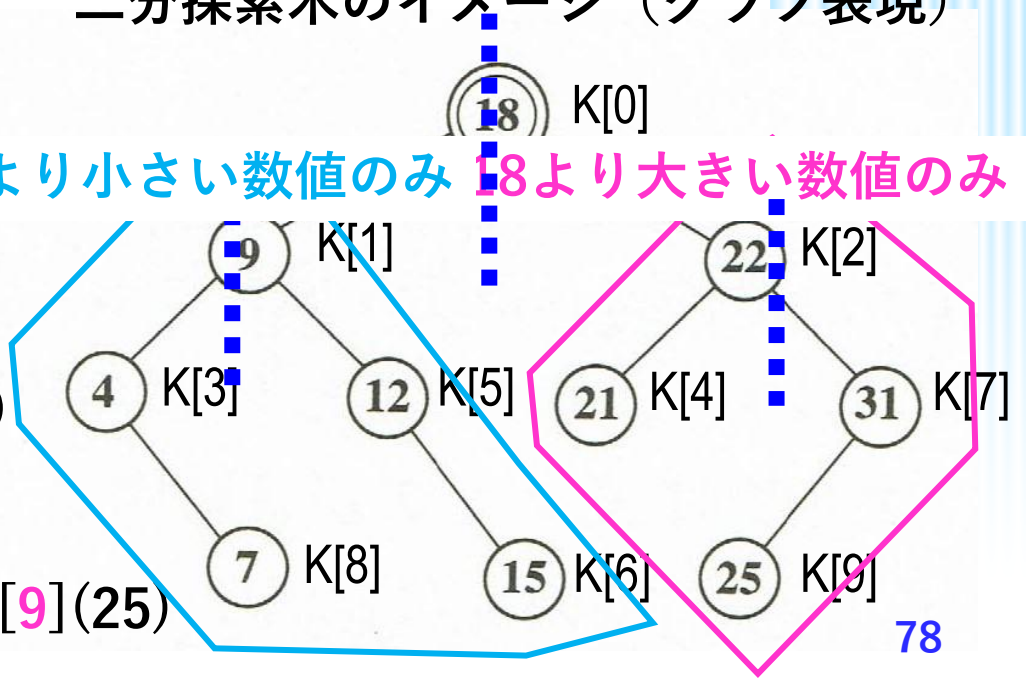
二分探索木の4本の1次元配列 K, P, L, R による実現

	0	1	2	3	4	5	6	7	8	9
(キー) K	18	9	22	4	21	12	15	31	7	25
(親) P	-1	0	0	1	2	1	5	2	3	7
(左子) L	1	3	4	-1	-1	-1	-1	9	-1	-1
(右子) R	2	5	7	8	-1	6	-1	-1	-1	-1

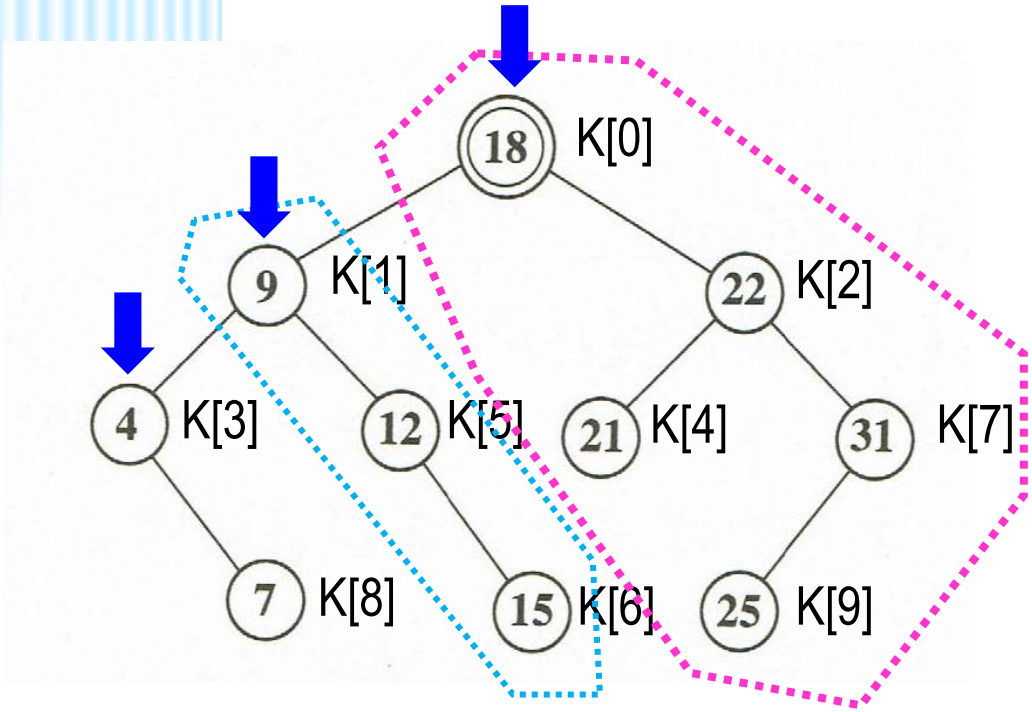
親(上方向)
K[1]の親はP[1](=0→K[0]=18)
K[4]の親はP[4](=2→K[2]=22`
左子(左下方向)
K[1]の左子はL[1](=3→K[3]=4)
右子(右下方向)
K[1]の右子はR[1]](=5→K[5]=12)

根：K[0](18)
葉：K[8](7),K[6](15), K[4](21), K[9](25)

二分探索木のイメージ（グラフ表現）



2.3 (平衡)二分探索木の利用(2)



	二分探索木
member(4)	3
member(15)	4
member(16)	4
member(22)	2
member(7)	4

- 最悪計算時間：要素数 10 のとき 4C　この 4 は？
  要素数　n のとき　xC　この x は？
  $x=(\lfloor \log_2 n \rfloor + 1)C$

2.4 $\log_2 n$ について

$2^x = n \iff x = \log_2 n$

切り捨て

x	$n=2^x$	$\log_2 n$	$\lfloor \log_2 n \rfloor + 1$
1	2	1	2
2	4	2	3
3	8	3	4
3.3	10	3.3	4
4	16	4	5
⋮	…	…	⋮
9.9	1000	9.9	10
10	$2^{10}=1024$	10	11
⋮	…	…	⋮
19.9	10^6	19.9	20
20	$2^{20}=1024^2$	20	21

通常は、 $\log_2 1000$ $(= \log_2 10^3)$ $\fallingdotseq$ $\log_2 1024=10$ として計算

3.1 最悪計算時間の比較

計算時間： データ構造の作成＋k回の探索時間

（１）初等的な場合：

1次元配列へのデータ入力＋k回の探索時間

（２）(平衡) 2 分探索木の場合：

2 分探索木の作成＋k回の探索時間

2 つのアルゴリズムの最悪計算時間の比較

	初等的な場合	平衡 2 分探索木に基づく場合
データ構造作成	Cn	$Cn + Cn\log_2 n$
k 回の member(x) の最悪計算時間	kCn	$kC(\lceil \log_2 n \rceil + 1)$
合計	$(k+1)Cn$	$Cn + Cn\log_2 n + kC\lceil \log_2 n \rceil + kC$

(平衡)2分探索木を使う利点：

- データ構造作成に少し時間がかかるが、(nが大きくなると)
- 探索回数 k が増えると全体の計算時間増加が抑えられる。

3.2 最悪計算時間の具体値(単位:秒)(1)

$C = 10^{-3}$ 秒, $n = 1000(= 10^3) \approx 2^{10}(= 1024)$ のとき
(ただし、$\lceil \log_2 10^3 \rceil \approx \log_2 2^{10} = 10$ とする)

	初等的な場合		平衡2分探索木に基づく場合
k	$k+1$		$0.011k + 11$
1	2	$<$	11.011
10	11		11.11
20	21	$\geq$	11.22
100	101		12.1
1000	1,001		22

3.2 最悪計算時間の具体値(単位:秒)(2)

$C = 10^{-6}$ 秒, $n = 1000(= 10^3) \approx 2^{10}(= 1024)$ のとき
(ただし、$\lceil \log_2 10^3 \rceil \approx \log_2 2^{10} = 10$ とする)

	初等的な場合		平衡2分探索木に基づく場合
k	$0.001k + 0.001$		$0.000011k + 0.011$
1	0.002	$<$	0.011011
10	0.011		0.01111
20	0.021	$\geq$	0.01122
100	0.101		0.0121
1000	1.001		0.022

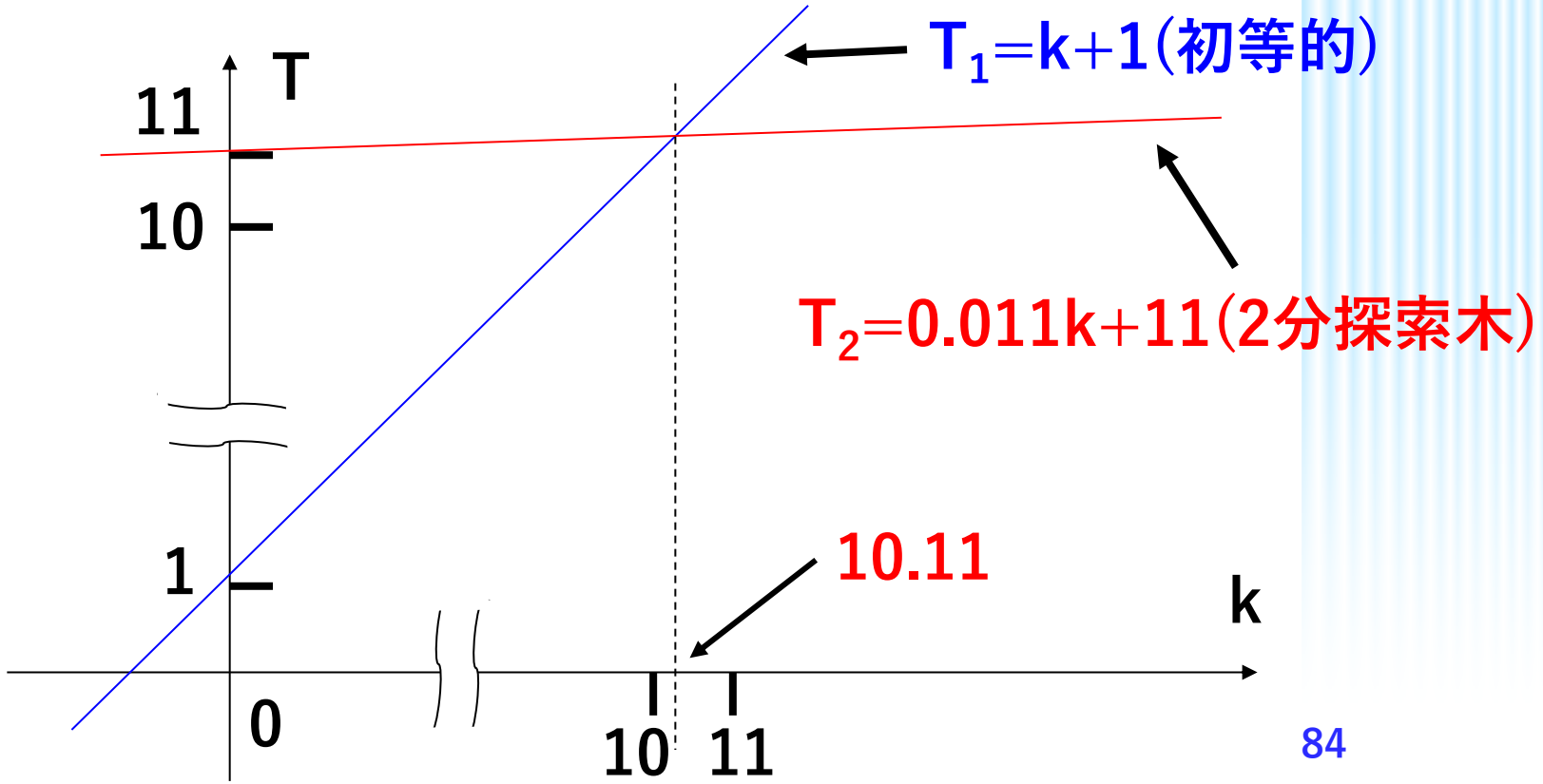
3.3 kの値による t_1 と t_2の比較

$t_1 = 0.001k + 0.001$ と $t_2 = 0.000011k + 0.011$ の大小比較

(実際は $T_1 = 10^3 t_1 = k + 1$, $T_2 = 10^3 t_2 = 0.011k + 11$ の比較)

$T_1 = T_2$ となる k の値：$k = 10.11$

$k < 10.11$ では $T_1 < T_2$; $k \geq 10.11$ では $T_1 \geq T_2$



3.5 コンピュータでのa+bの計算（プログラミングと内部処理の具体例）

- 2数の和a+bを求めるＣ言語プログラム（a=45, b=27のとき）を例として、実行過程を説明する。

コンピュータ内部で処理の概要

説明のポイント

- 加算a+bの動作について
- ソースプログラム（人間による記述）
- アセンブラ表現と機械語について（コンピュータ処理用のデータ）
- 加算の内部での実際の動作

3.5.1 2数の和a+bを求める C言語プログラム（a=45, b=27の例）

```
/* Program name: sum.c */
/* Sum of two integers */
#include <stdio.h>
main()
{
        int a=45, b=27, c;
        c=a+b;
}
```

3.5.2 sum.cのコンピュータ内部の姿(1)

	8	4	2	1	8	4	2	1	16進	アセンブリ言語での記述
1	0	1	0	1	0	1	0	1	55	pushl %ebp
2	1	0	0	0	1	0	0	1	89	movl %esp, %ebp
3	1	1	1	0	0	1	0	1	e5	
4	1	0	0	0	0	0	1	1	83	subl $12, %esp
5	1	1	1	0	1	1	0	0	ec	(subl $0xc, %esp)
6	0	0	0	0	1	1	0	0	0c	
7	1	1	0	0	0	1	1	1	c7	movl $45, -4(%ebp)
8	0	1	0	0	0	1	0	1	45	45をaのスタックに格納
9	1	1	1	1	1	1	0	0	fc	
10	0	0	1	0	1	1	0	1	2d	
11	0	0	0	0	0	0	0	0	00	
12	0	0	0	0	0	0	0	0	00	
13	0	0	0	0	0	0	0	0	00	

レジスタAとスタックを使う準備

3.5.2 sum.cのコンピュータ内部の姿(2)

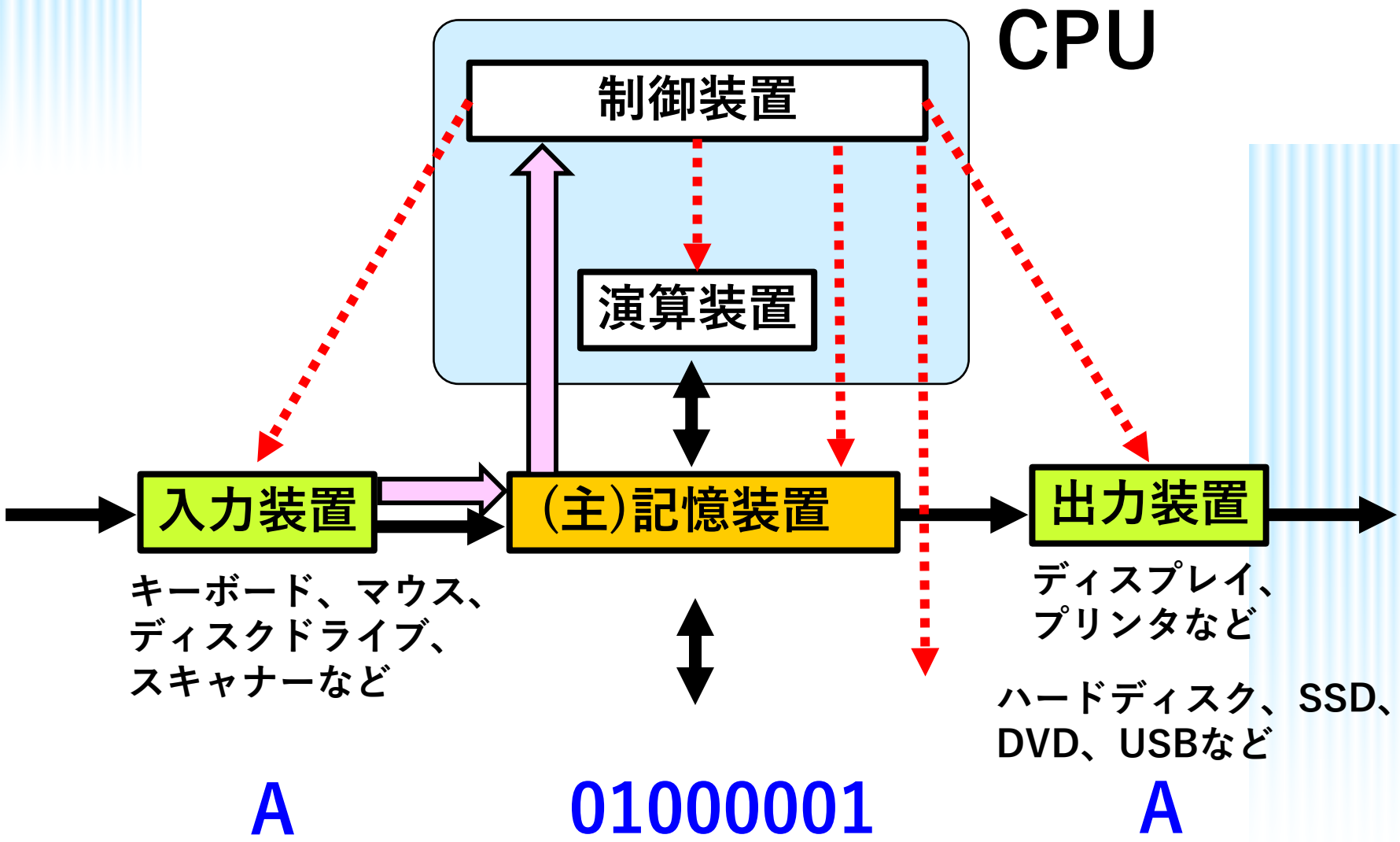
	8	4	2	1	8	4	2	1		アセンブリ言語での記述
14	1	1	0	0	0	1	1	1	c7	movl $27, -8(%ebp)
15	0	1	0	0	0	1	0	1	45	(movl $0x1b, 0xfffffff8(%ebp))
16	1	1	1	1	1	0	0	0	f8	27をbのスタックに格納
17	0	0	0	1	1	0	1	1	1b	
18	0	0	0	0	0	0	0	0	00	
19	0	0	0	0	0	0	0	0	00	
20	0	0	0	0	0	0	0	0	00	(加算の操作)
21	1	0	0	0	1	0	1	1	8b	movl -4(%ebp), %eax
22	0	1	0	0	0	1	0	1	45	45をレジスタAに移動
23	1	1	1	1	1	1	0	0	fc	

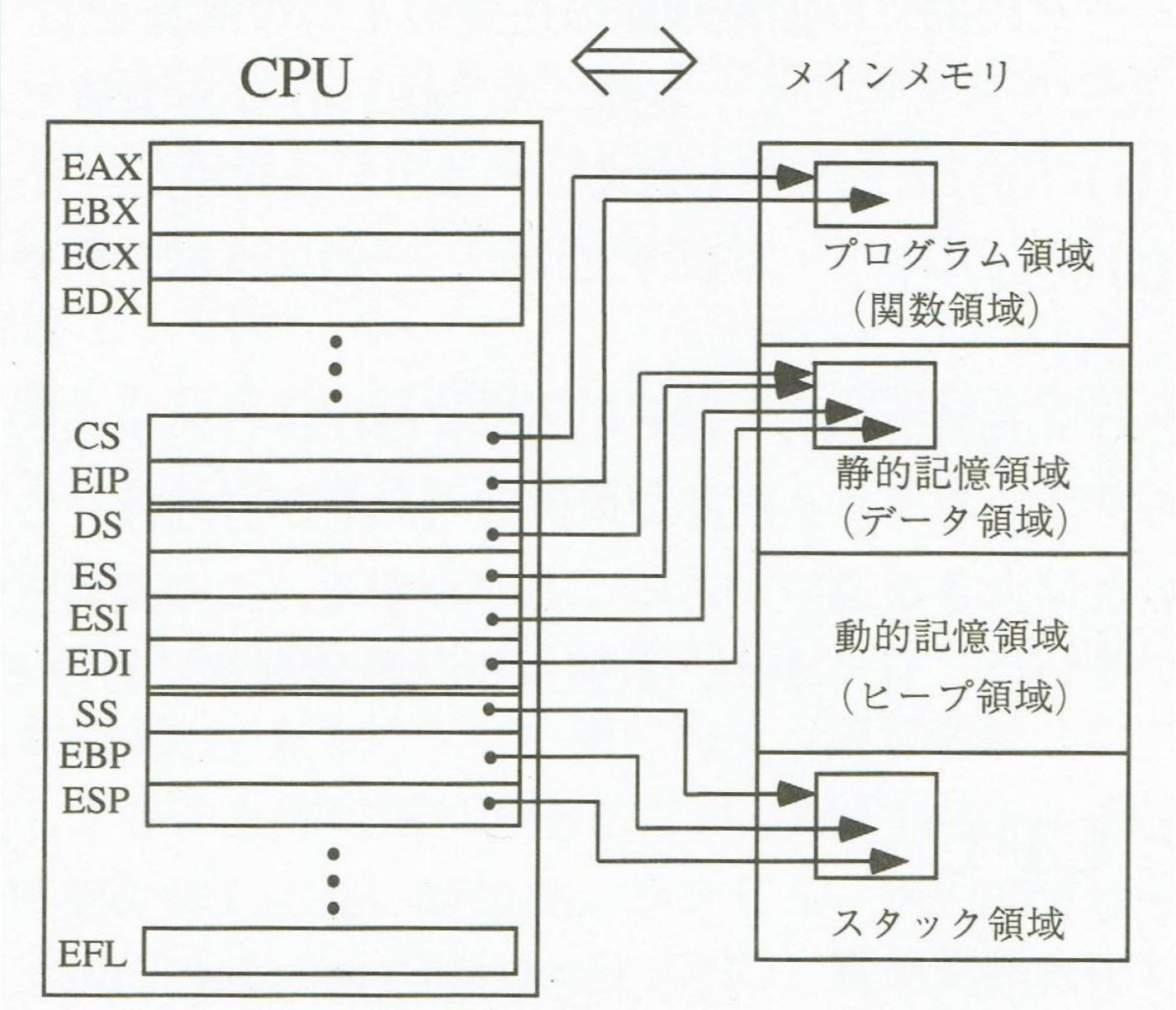
3.5.3 sum.cのコンピュータ内部の姿(3)

	8	4	2	1	8	4	2	1		アセンブリ言語での記述
24	0	0	0	0	0	0	1	1	03	addl -8(%ebp), %eax
25	0	1	0	0	0	1	0	1	45	27をレジスタAに加える
26	1	1	1	1	1	0	0	0	f8	
27	1	0	0	0	1	0	0	1	89	movl %eax, -12(%ebp)
28	0	1	0	0	0	1	0	1	45	72をcのスタックに格納
29	1	1	1	1	0	1	0	0	f4	
30	1	1	0	0	1	0	0	1	c9	leave
31	1	1	0	0	0	0	1	1	c3	ret

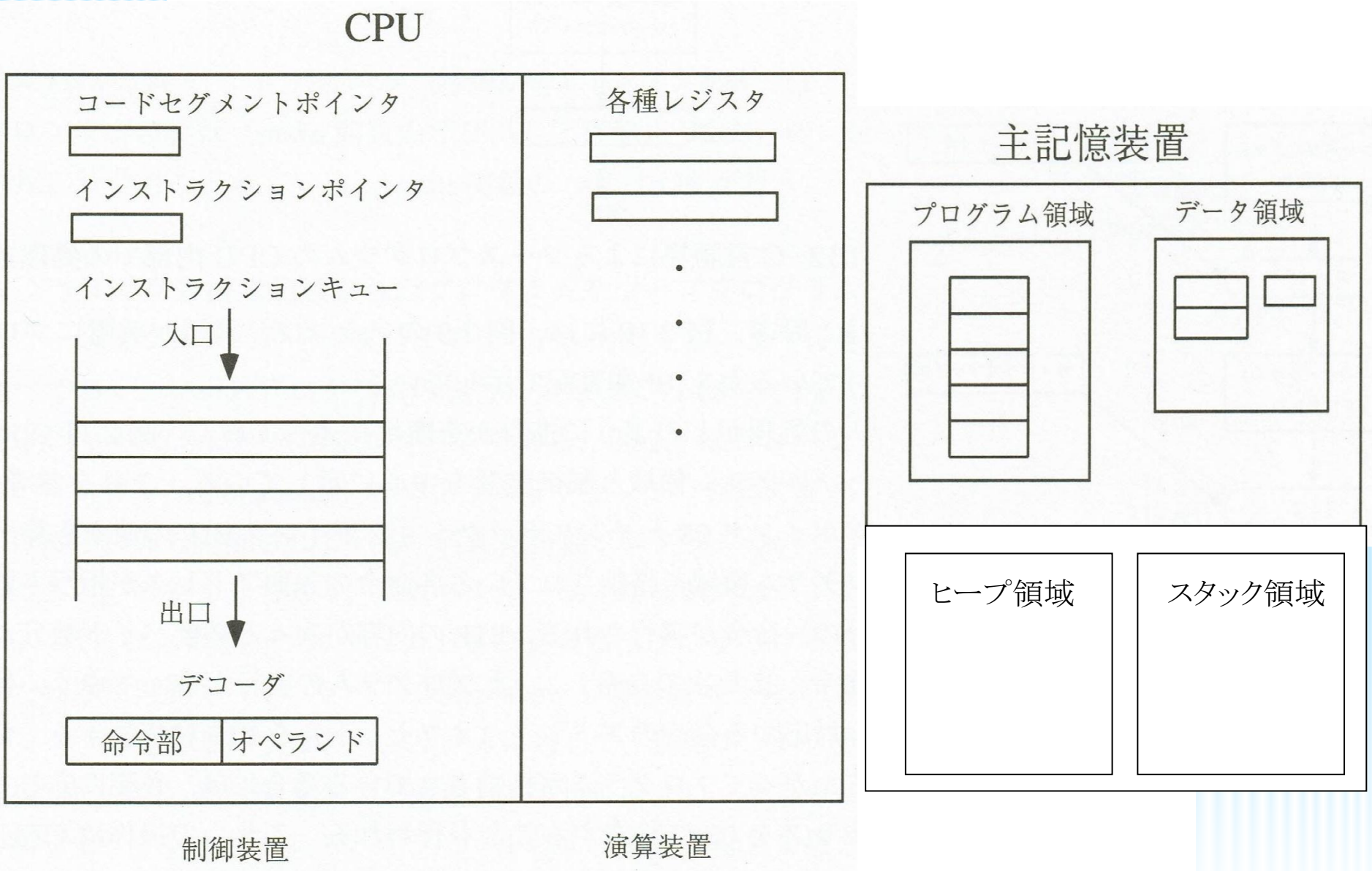
終了操作

コンピュータの仕組み（再掲）





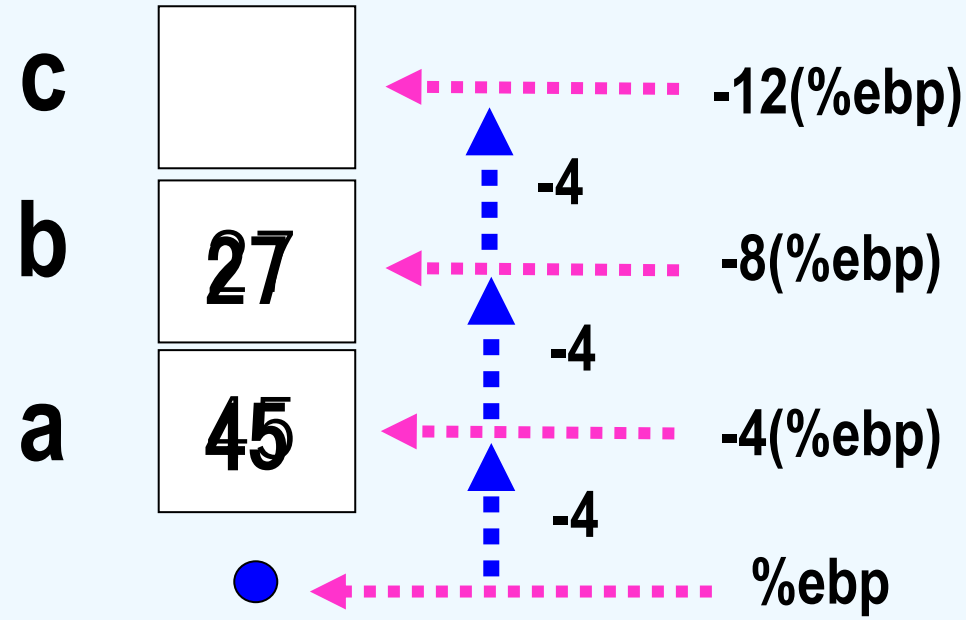
メインメモリとCPUの概念図
(制御装置、入出力装置、周辺装置は省略)



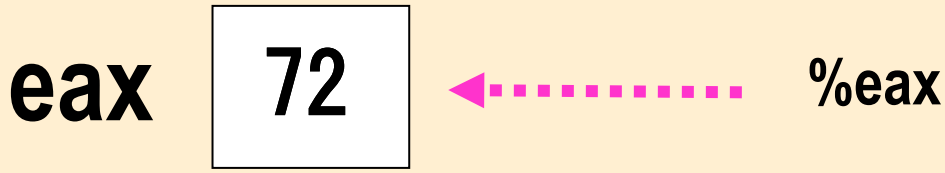
プログラム領域と制御装置の概要

3.5.4 sum.cの実際の動作(1)

主記憶内のスタック



スタック領域

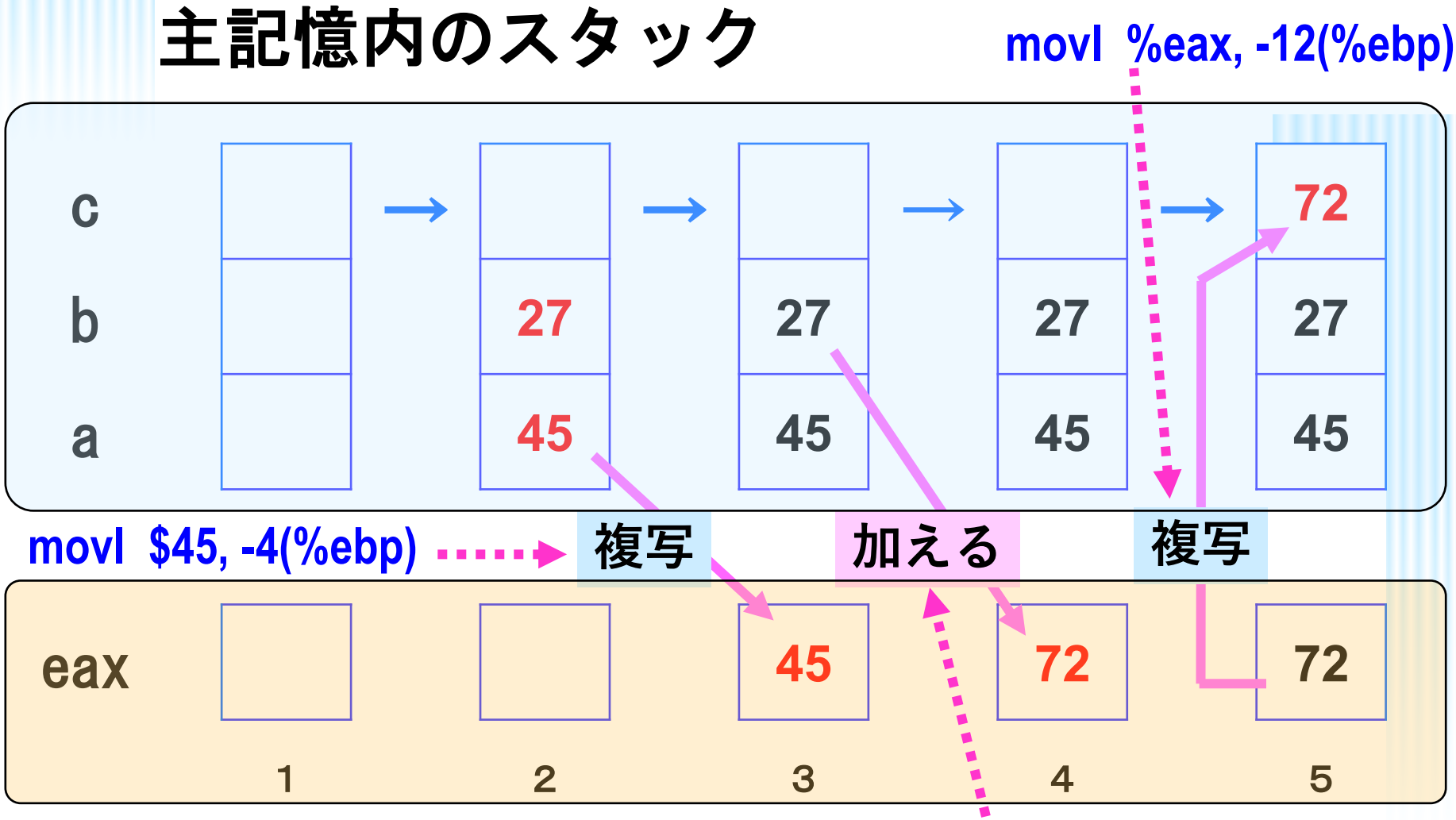


CPU

CPUのレジスタ

3.5.4 sum.cの実際の動作(2)

主記憶内のスタック



講義後半のまとめ

- 以下の二つの話題により、アルゴリズムやデータ構造の工夫によって処理時間や使用メモリ量の削減が実現できることを説明。
- ホーナー法により、多項式計算の計算時間、使用メモリ量がともに減少できることを具体例に基づいて説明。
- 二分探索木の利用により、探索の最悪時間を減少させることができることを具体例を用いて説明。
- プログラミングと内部処理の具体例

  コンピュータ内部でのa+bの実際の計算動作を紹介。

講義予定内容一覧(1)　　：今回は省略の予定

1. コンピュータの素顔は？
(イントロダクションに代えて)
    1.1　コンピュータは魔法使い？
    1.2　コンピュータは台所とお母さん
    1.3　現代社会でのコンピュータの役割
    1.4　コンピュータの素顔
2. コンピュータの機能（ハードウェア）と情報処理の仕組み（ソフトウェア）
    2.1 0と１だけの世界に生きる
    2.2 数体系：数値の扱いの基本
        2.2.1 基数と桁
        2.2.2 大きさの対応
        2.2.3 10進数 - 16進数（2進数）の表記
        2.2.4 整数 「10進→2進→8進,16進」 なる変換
        2.2.5 正整数の 「10進→2進→16進」 なる変換
        2.2.6 小数点数の「10進→２進」なる変換

講義予定内容一覧(2)　　：今回は省略の予定

- 2.3 コンピュータの仕組と内部データ
  - 2.3.1 内部データ（符号化）
  - 2.3.2 コンピュータ内部の文字データ
  - 2.3.3 文字コードと符号化文字集合
  - 2.3.4 (半角)文字、数字、記号、等
  - 2.3.5 コンピュータでの数値の扱いと誤差
- 2.4 0.1を100個加えると？
  - 2.4.1 0.1を100個加えると
  - 2.4.2 0.1の内部での姿（32ビット版）
  - 2.4.3 数値は誤差がいっぱい
  - 2.4.4 0.1, 0.3, 0.5を各々100個加えた値
  - 2.4.5 0.3を100個加える過程を見ると
  - 2.4.6 数値計算は誤差に注意を
- 講義前半のまとめ

講義予定内容一覧(3)

3. 情報の収納方法（データ構造）や処理手順（アルゴリズム）
   - 3.1 アルゴリズムとは
   - 3.2 コンピュータを直接に操る
   - 3.3 アルゴリズムデザイン
     - 3.3.1 アルゴリズム設計の実際
     - 3.3.2 方法１（初等的計算法）
     - 3.3.3 方法２（ホーナー法）
     - 3.2.4 方法１と方法２の比較
     - 3.3.5 例題１の計算
     - 3.3.6 別の例題の計算
     - 3.3.7 例題１の計算
     - 3.3.8 別の例題の計算
     - 3.3.9 例題１、例題2の共通性

講義予定内容一覧(4)

：今回は省略の予定

3.4 アルゴリズムの評価について
- 3.4.1 アルゴリズムの評価基準
- 3.4.2 いろいろな基本的アルゴリズム
- 3.4.3 探索アルゴリズムによる説明
- 3.4.4 最悪計算時間
- 3.4.5 二分探索木の利用
- 3.4.6 $\log_2 n$ について
- 3.4.7 最悪計算時間の比較
  - 3.4.7-1 最悪計算時間の具体値(秒)
  - 3.4.7-2 最悪計算時間の具体値(秒)
  - 3.4.7-3 kの値による t_1 と t_2の比較

3.5 コンピュータでのa+bの計算
- 3.5.1 2数の和a+bを求めるC言語プログラム（a=45, b=27の例）
- 3.5.2 sum.cのコンピュータ内部の姿
- 3.5.3 sum.cの実際の動作

講義後半のまとめ