1. コンピュータの基礎知識

1.1. コンピュータの基本構成

5 大要素

○制御装置 ：必要な指示を出し全体の動きを “制御” します。

○演算装置 ：指示通り、“演算(計算)” を行います。

○入力装置 ：情報を “入力” します。

○出力装置 ：情報を “出力” します。

○記憶装置 ：情報を “記憶” します。

1.1.1. ハードウェアとソフトウェア

○ハードウェア 形があり、触れるもの
パソコン本体、キーボード、ジョイスティック、USB メモリなど
ただし、プログラムが印刷された紙などはハードウェアとは呼びません。

○ソフトウェア ルールに従って構成されるコンピュータへの指示そのもの。
時には、発想そのものもソフトウェアと呼ぶ場合もあります。

1.2. コンピュータでの情報表現

1.2.1. 現在のコンピュータが理解できる情報

現在のコンピュータは、定められた電圧と比較し、電圧がそれよりも高いか低いかという情報しか扱うことができない。

→ 電圧の高低の組み合わせで様々な状態を表現する。
電圧の高低の情報 1 つ・・・・・・・・・・・・・・・ビット(bit)
電圧の高低の情報を 8 個まとめたもの・・・バイト(Byte)

※超重要！
まとめて扱う数が増えると、表現できるパターンが多くなる。

n ビットを使うと、$2^n = 2 \times 2 \times 2 \times \cdots\cdots$
2 を n 回掛ける

1.2.2. 文字の表現

bit の組み合わせに文字をあてはめて表現する。 組み合わせ表 → 文字コード

代表的な文字コード

○ASCII(アスキー) 7bit で文字を表現している。その他の文字コードの基本。

○EUC Unix で利用されていた、日本語も利用できる文字コード
英数文字は 1 バイト、日本語は 2 バイトで表現する。

○Shift-JIS(シフト・ジス) マイクロソフトや日本の企業により策定された

○Unicode(ユニコード) 世界中の文字をひとつの文字コードで表現しようとした。現在は UTF-8/UTF-16 が主流

1.3. 補助単位

1.3.1. 大きな桁の補助単位

大きさ	10 の乗数	記号	読み
1,000	3	K	キロ
1,000,000	6	M	メガ
1,000,000,000	9	G	ギガ
1,000,000,000,000	12	T	テラ
1,000,000,000,000,000	15	P	ペタ
1,000,000,000,000,000,000	18	E	エクサ
1,000,000,000,000,000,000,000	21	Z	ゼタ
1,000,000,000,000,000,000,000,000	24	Y	ヨタ

1.3.2. 小さな桁の補助単位

大きさ	10 の乗数	記号	読み
0.001	-3	m	ミリ
0.000001	-6	μ	マイクロ
0.000000001	-9	n	ナノ
0.000000000001	-12	p	ピコ
0.000000000000001	-15	f	フェムト
0.000000000000000001	-18	a	アト
0.000000000000000000001	-21	z	ゼプト
0.000000000000000000000001	-24	y	ヨクト

1.4. 各種装置の分類

1.4.1. 主記憶装置と補助記憶装置

検定においては下記のイメージで大丈夫

主記憶装置 ・・・・ メモリ : 速い、高い、少ない

補助記憶装置 ・・・ ハードディスク : 遅い、安い、大きい

1.4.2. 入出力装置

最近ではあまり出題されない

入力装置 : マウス、キーボード、

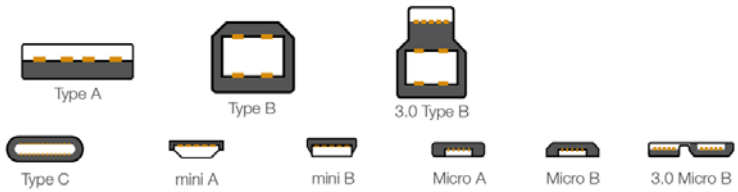
出力装置 : ディスプレイ、プリンタ

1.4.3. インターフェイス

近年出題傾向が高くなっている

1.4.3.1. USB(Universal Serial Bus)

コンピュータ等の情報機器に周辺機器を接続するため、最も広く利用されているシリアルバス規格の 1 つ、現在電源供給の規格として USB-PD(ユーエスビー・ピーディー)、裏表なしのコネクタになった USB-C がある。コネクタ形状は下記の通り。

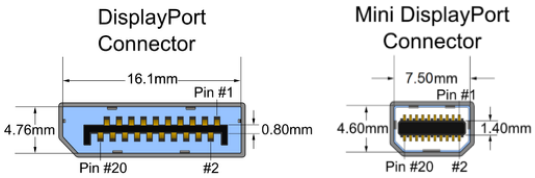


速度、給電規格

バージョン	USB 1.0,1.1	USB 2.0	USB 3.0	USB 3.1	USB 3.2	USB-C	USB -PD
速度	12Mbps	480Mbps	5Gbps	10Gbps	20Gbps		
最大 給電能力	500mA 2.5W	500mA 2.5W	900mA 4.5W	1A	1A	3A 15W	5A 100W

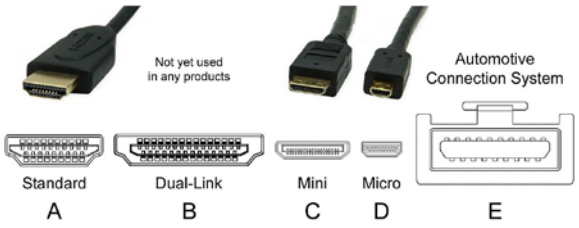
1.4.3.2. ディスプレイポート

映像出力インタフェースの規格で、高解像度のディスプレイへの対応を当初から視野に入れていたため、医療分野では広く利用されている。



1.4.3.3. HDMI（High-Definition Multimedia Interface）

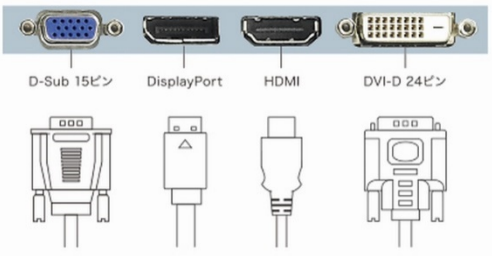
HDMI はデジタル家電向けのインタフェースで、2002 年 12 月に HDMI 1.0 の仕様が策定された。最新規格は HDMI 2.1。PC とディスプレイの接続標準規格である DVI を基に、音声伝送機能や著作権保護機能など AV 家電向けに改良したものである。



1.4.3.4. D-sub（VGA 端子）

古くから広く利用されている規格でプロジェクタや古いディスプレイで利用されている。、高解像度に対応できないため、今後の利用は望めない。

左に、ディスプレイの入力端子の例を挙げておく



1.5. プログラムの作成から実行まで

コンピュータは、最終的に 0/1 の情報しか扱えない。そのため、人間の作成するプログラムを、コンピュータが理解できる情報に変換する必要がある。

1.5.1. プログラムが動作するまでの流れ

①コーディング/プログラミング

言語仕様に沿って人間が、プログラムを作成する。

この段階の情報は “英数文字/記号” のテキストデータ ← プログラム編集ソフト “テキストエディタ”

②コンパイル

テキストデータであるプログラムを解析し、コンピュータが理解できる情報に翻訳/変換する。

③リンカー

定型の処理など、ライブラリより組み込む処理

④実行可能プログラム

実行できるように変換されたプログラム。

1.5.1.1. IDE 環境

上記の機能に加え、デバッグ機能など統合された開発環境、Visual Studio 2017 も IDE のひとつです。

1.6. ネットワークの基礎知識

1.6.1. LAN(Local Area Network)

建物内など限定された範囲のネットワーク環境、しかし近年仮想的な LAN で都市間を結ぶこともあり、物理的な距離ではなく、利用者のお互いの関係性が高いネットワークを表すようになっている。

1.6.2. WAN(Wide Area Network)

LAN に対して都市間など広範囲なネットワーク。

1.6.3. 通信速度

ネットワークでは通信速度を bps(bit/second ：ビット・パー・セカンド)で表し、1 秒間で通信できる bit 数で表現する。

※Byte 数ではない。Byte 変換時には 8 で割る必要がある

以下に関しては、別件での学習あり、今回概要説明のみ

1.7. ファイルとデータベース

1.8. コンピュータの種類

2. 数値の表現

2.1. 基数変換

基数とは、「桁上がり」ということを考えればよい。10 進数であれば、0～9 の 10 種類の数字を利用する。そのため 9 以上増えると桁上がりが発生する。2 進数は 0 と 1 のふたつ、16 進数は 0～F までの 16 の数字を用いる。

2.2. 2 進数と 10 進数と 16 進数の関係

4bit で 2^4 の組み合わせが可能 → 0～15 までの 16 通りの数字に対応させることができる

2 進数	10 進数	16 進数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

← なぜ 2 進数 0101 が 10 進数 5 になるのか？

$0\times2^3+1\times2^2+0\times2^1+1\times2^0$

$=0\times8 +1\times4 +0\times2 + 1\times1 = 5$

2 進数なので基数は “2“

同様に $(1110)_2$ についても考えてみる

$1\times2^3+1\times2^2+1\times2^1+0\times2^0$

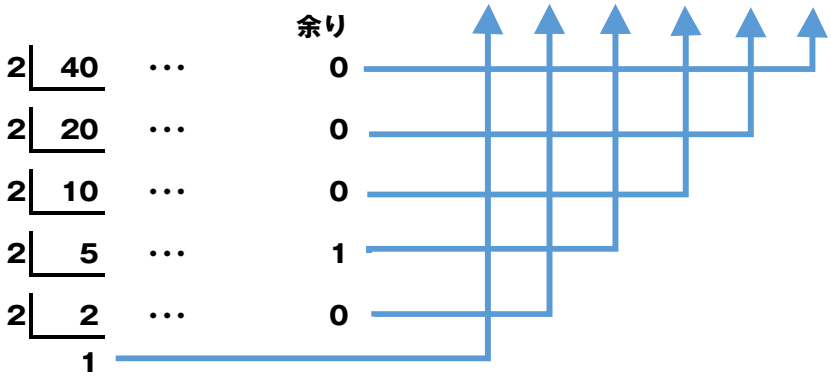
$= 1\times8 +1\times4 + 1\times2 + 0\times1 = (14)_{10}$ また 16 進では $(E)_{16}$

2.3. 10 進数から 2 進数への変換：小数点以上編

10 進数から 2 進数への変換方法は、対象の数字を基数である“2”で割って、その余りを並べるだけ！

10進数40を2進数に変換する。 ・・> 1 0 1 0 0 0

※2で割る！



※8 進数、16 進数は割る数がそれぞれの基数に変わるだけで要領は同じです。

2.4. 小数点以下の場合

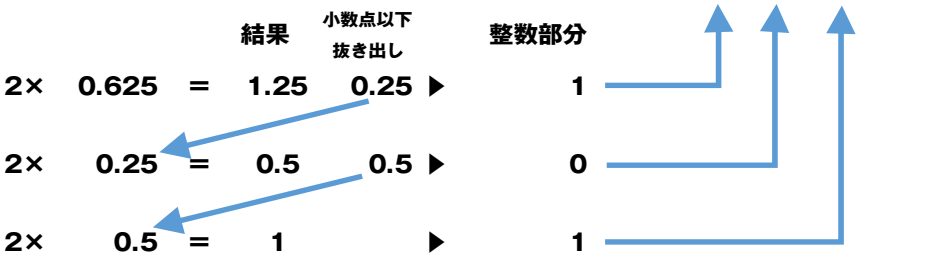
$(0.101)_2$ はどのようになるのか？

$$(0.101)_2 = 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$
$$= 1\times1/2 + 0\times1/4 + 1\times1/8$$
$$= 1\times0.5 + 0\times0.25 + 1\times0.125 = (0.625)_{10}\text{となる}$$

2.4.1. 10 進数から 2 進数の変換：小数点以下編

10進数 0.625 を2進数に変換する。 ・・> (0. 1 0 1) 2

※2を掛けて、整数部分を抜き出す！



※8 進数、16 進数は割る数がそれぞれの基数に変わるだけで要領は同じです。
小数点のある数字は、小数点以上と、小数点以下で分けて計算し後で合計します。

いったん 2 進数に変換することで、16 進への変換も楽になります。

2.5. 2 進数の計算

2 進数同士の計算は桁上がりに注意するだけ。

2進数の加算は下記のパターンのみ。

```
  0     0     1     1
+ 0   + 1   + 0   + 1
---   ---   ---   ---
  0     1     1    10
```

教科書の問題では、異なる基数の数字の計算を行っているが、まずは 2 進数で慣れていこう！

減算処理(引き算)はあまり重要ではない。

2.6. その他の数値表現

下記の表現は基本情報で出題されるが、演算に不向きで近年での利用は少ないため、用語の確認だけで良い

2.6.1. BCD コード

教科書 p5 の表記通りに、4bit で 0-9 までの数値を表す方法

2.6.2. ゾーン 10 進表記

一桁の数値を 1 バイトで表す方法で下記の特徴を持つ

〇 1 バイトの上位 4 ビットをゾーン部と言い、JIS コードの場合は 0011、EBCDIC コードの場合は 1111 が設定される

〇 1 バイトの下位 4 ビットが数値を表す

〇最下位桁のゾーン部は正負の符号を表す(正:1100、負 1101 とすることが多い)

2.6.3. パック 10 進表記

ゾーン 10 進と同様に 1 桁の数値を 4 ビットで表現する方法で下記の特徴を持つ

〇下位の 4 ビットは正負の符号を示す(正:1100、負 1101 とすることが多い)

〇桁数か偶数のときは、左の左端に 0000 を入れ、データを整数バイトで表せるようにする

2 進数での負の数の表現：補数

2.7. 補数とは目標の数になるために、必要な数のこと

7 の 10 の補数･･･3

7 の 9 の補数････2　　実は小学校で勉強してます。

コンピュータ情報処理では、この “補数” を使うと 2 進数の “負の数” が表現できる。

2.8. 2 の補数を使った計算

$(3)_{10}$ − $(1)_{10}$ の計算をしてみます。

$(3)_{10}$ ＝ $(0011)_2$　　$(1)_{10}$= $(0001)_2$　なので答えは $(2)_{10}$ =$(0010)_2$ になってほしい！

$(1)_{10}$= $(0001)_2$ の **2 の補数** ：**ひっくり返して 1 を足す**

･･･････････$(1110)_2$ + $(0001)_2$ = $(1111)_2$

$(3)_{10}$ + (-$(1)_{10}$) = $(0011)_2$ + $(1111)_2$ = $(0010)_2$ =$(2)_{10}$

※便宜上 2 進数を 4 桁で表現していますが、3 桁でも 16 桁でも結果は変わりません。試してみてください。

2.9. 2 の補数で表す負の数

2 進数	10 進数		2 の補数	10 進負数		
0000	0	+	0000	0	=	0
0001	1	+	1111	-1	=	0
0010	2	+	1110	-2	=	0
0011	3	+	1101	-3	=	0
0100	4	+	1100	-4	=	0
0101	5	+	1011	-5	=	0
0110	6	+	1010	-6	=	0
0111	7	+	1001	-7	=	0
			1000	-8		

これだと、 ＋0，－0 のようなダブりもないし、負の数は計算前に 2 の補数にすれば、加算回路だけで加減算が実装できる。

2 の補数で負の数を表現する場合「常に負の数の方が表現範囲がひとつ大きい」という特徴を持つ。

2.10. 小数点表現

コンピュータは、小数などの実数、文字などは直接扱えない！

小数点の表現は、決められた範囲内で工夫した結果です。 ・・・・ パソコンは手順通り処理するのみ。

2.10.1. 固定小数点表現

国家試験ではほぼ出題されません。

また、出題されても小数点位置は最も右に設定されるため普通の 2 進数の問題として考えてください。

ポイント)

〇数値の表現範囲（最大値、最小値）

2.10.2. 浮動小数点表現

午後の設問で出題されることが比較的多い。しかもこのまま出題される。

ポイントは下記の通り

- 〇 指数表現
- 〇 正規化 ：正規化の方法にもいくつか方法がある
- 〇 S（符号）
- 〇 E（指数部） ： 形式として 「**バイアス方式**」、「2 の補数表現」 があります。
- 〇 M（仮数部） ： 仮数の正規化方法にも、正規化後の整数一桁目が “0” と “1” があります。
- 〇 **表現方法に、いくつかバージョンがある ・・・ 過去問解く際に混乱するポイント 条件をよく読むこと**
  代表的な形式 IEEE754

<データフォーマット>

※IEEE754(32bit 単精度)の場合

符号	指数	仮数

① 基数は 2 とする

② 正規化する IEEE754 では正規化後の整数 1 桁目は “1” ・・・ **1**.1010・・

③ 符号部：1 ビット 負：1 、 正：0

④ 指数部：8 ビット、乗数は下駄履き表現(バイアス表現：127 加算) ※教科書では 4 ビット、2 の補数表現です。

⑤ 仮数部：23 ビット ※教科書では 11 ビット。

条件さえしっかり押さえていれば、大丈夫！

※IEEE アイ・トリプル・イー アイ・イー・イー・イーではありません。IEEE とは、電気・電子分野における世界最大の専門化組織。主に工学分野における学会としての活動と、工業技術の標準化団体としての活動を行っています。

2.10.3. バイアス方式(下駄履き方式)

バイアス方式とは、浮動小数点形式の指数の表現について 2 の補数を利用せず、P5 の表を単純に上下に分けて上を正の数、下側を負の数として表現する方式です。

教科書では、元の数字に指定された数字を加算する。とよく書かれていますが、最終的に 0～最大値(この場合 111)に負の数を当てはめるために、指定された数字を減算すると考えれば、わかりやすいと思います。

(3bit 10進数で 0～7)			
2進数	10進数	下駄履き表現	
000	0	- 4=	-4
001	1	- 4=	-3
010	2	- 4=	-2
011	3	- 4=	-1
100	4	- 4=	0
101	5	- 4=	1
110	6	- 4=	2
111	7	- 4=	3

2.11. シフト演算

シフトは、数字の桁をずらすことで、ずらすことを “シフトする” といいます。桁をずらすだけなのですが、下記のような特徴があるため、計算に利用できます。

分かりやすいように 10 進数で考えた場合下記の通りになります。ずらすだけで、10 倍や 1/10 倍になりますね、

右に 1 シフト	1	1/10 になってます。 ↑
	10	ここを基準とします
左に 1 シフト	100	↓ 10 倍になっています

コンピュータは 2 進数なので下記の通りになります。

		10 進表記	
右に 1bit シフト	001	1	1/2 になってます。 ↑
	010	2	ここを基準とします
左に 1bit シフト	100	4	↓ 2 倍になっています

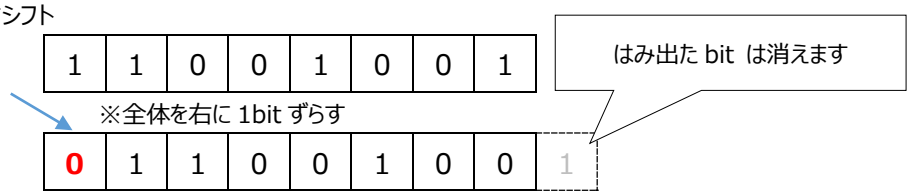
例えば、5 倍する場合、左に 2bit シフト(2×2=4 倍)したものに、元の数を加算すれば 5 倍の計算ができます。

2.11.1. 論理シフト

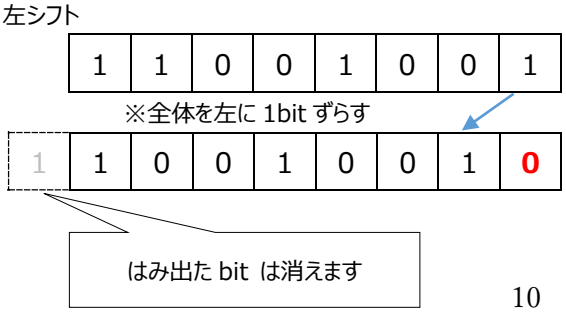
左右にシフトした場合、空いた場所に **0** が入ります。

例)

1bit 右シフト



1bit 左シフト

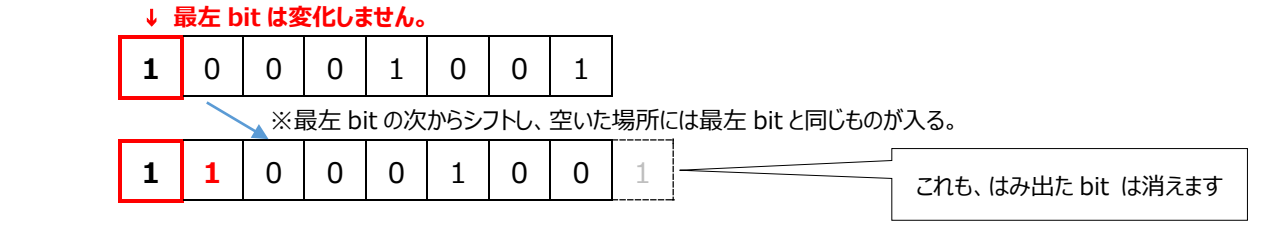


2.11.2. 算術シフト

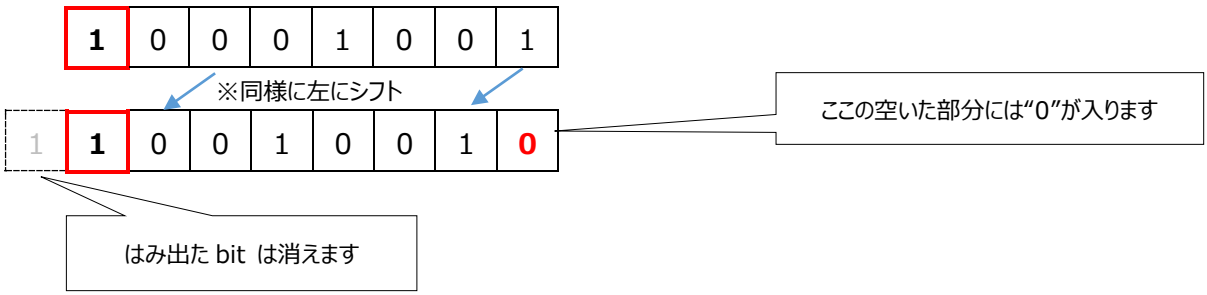
論理シフトと違って、計算を前提にするため最左ビットを符号ビットとして扱います。**そのため最左ビットの内容は、算術シフトでは変化しません。**

例)

1bit 右シフト



1bit 左シフト

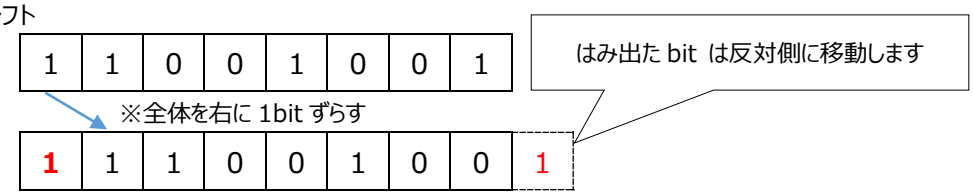


2.12. 循環シフト

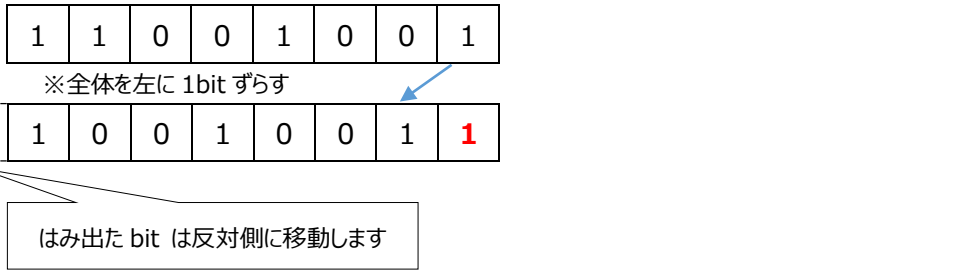
左右にシフトした場合、空いた場所に**反対側からあふれた bit** が入ります。

例)

1bit 右シフト



1bit 左シフト



2.13. 誤差

教科書 P76 の用語 「情報落ち」、「桁落ち」、「丸め誤差」確認

3. コンピュータ構成要素

3.1. プロセッサ

重要単語

「アドレス」、「レジスタ」、「再配置可能：リロケータブル」

3.2. アドレス指定方式

ポイントは、場所を指定する「アドレス」と、アドレスに格納されている「データ」を混同しないようにすること。

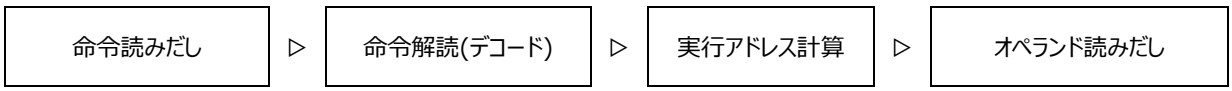
各種アドレス指定方式の違いは、**「アドレスをどのような方法で指定するか」**です。このように多くの方式があるのは、特許的な問題や、メーカーポリシー、作成された状況により、CPU の構成(アーキテクチャー)が異なるためです。

教科書 P86～88 参照

○直接アドレス指定方式
○インデックス修飾指定方式
○相対アドレス指定方式
○レジスタアドレス指定方式
○間接アドレス指定方式
○ベースレジスタ指定方式
○即値アドレス指定方式(イミディエイトアドレス指定方式)

3.3. 命令の実行手順

下記の命令の実行手順をとりあえず押さえましょう ← これが高速化技術にもつながります。



3.4. MIPS

1 秒間に命令を何百万回(100 万 = 10^6)実行できるかで性能を比較したもの。

3.5. 命令ミックスの計算

これも MIPS 同様 1 秒間に何回命令が実行できるかを計算するものです。

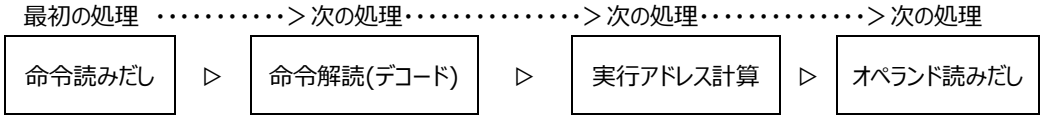
1 命令に数クロック必要で、その必要クロック数も命令により異なるため命令により処理時間の差が出ます。その命令の実行時間を平均したものを求めているだけです。

CPI ： 1 命令に必要なクロック数

1 クロックの時間 × CPI：必要クロック数 = 命令実行時間

3.6. 高速化技術

上記 3．3 にあるように命令実行にはいくつかの段階(ステージ)が必要です。1 命令づつ処理していくと、最初の「命令読み出し」から、「オペランド読みだし」まで 4 ステージかかってしまいます。



なので空いたところにどんどん命令を詰め込んでいこうとする方式がパイプライン処理になります。

この方法の、メリットとしてほとんど回路構成を変えずに、お手軽に高速化が可能になると言うことですが、デメリットもあります。

3.7. 論理演算

今まで、10 進数や 2 進数などを勉強してきましたが、ここでは 「真:true (トゥルー)」と「偽:false (フォールス)」 の計算をします。なので、計算結果は 「真」 か 「偽」 のどちらかになります。そのため入力は 2 つしか無い場合が多いです。

※混乱ポイント

ここでの真理値表では 真=1 、 偽=0 と表現されていますが、言語の実装で扱いが若干異なります。

3.7.1. 論理和（OR） ：論理的な足し算

論理和は、ひとつでも「真」 があれば OK として 「真」 を出力します。

入力 A	入力 B		出力
0	0	▷	0
0	1	▷	1
1	0	▷	1
1	1	▷	1

3.7.2. 論理積（AND） ：論理的なかけ算

論理積は、ひとつでも「偽」 があれば NG として 「偽」 を出力します。

入力 A	入力 B		出力
0	0	▷	0
0	1	▷	0
1	0	▷	0
1	1	▷	1

3.7.3. 排他的論理和(XOR,EOR) ：はいたてき論理和

入力の値が異なっているときのみ、OK として「真」を出力します。

入力 A	入力 B		出力
0	0	▷	0
0	1	▷	1
1	0	▷	1
1	1	▷	0

3.7.4. 否定

否定は、今の状態を反転させる

3.7.5. 論理演算の式の変形

教科書 P103 にある表ですが、これを知っていると数式を簡単に変形することができます。

論理和	論理積	否定
1) $0+A=A$	5) $0\cdot A=0$	9) $A\cdot\overline{A}=0$
2) $1+A=1$	2) $1\cdot A=A$	
3) $A+A=A$	3) $A\cdot A=A$	
4) $A+\overline{A}=1$	4) $A\cdot\overline{A}=0$	

3.7.6. ドモルガンの定理

特に必須ではありませんが、下記の関係を覚えておくと、計算が楽になります。教科書はこの式が正しいことを確認しています。

$$\overline{A+B}=\overline{A}\cdot\overline{B} \qquad \overline{A\cdot B}=\overline{A}+\overline{B}$$

3.8. ビットマスク

ステータス、条件をできるだけ小さいサイズに格納するためにフラグを用います。そのフラグに入っている情報を取り出す方法になります。

下記の場合ビット列は $(\ 1001\ 0011\ )_2 = (93)_{16}$ となります。

7	6	5	4	3	2	1	0
毒	猛毒	麻痺	眠り	無敵	アイテム 1	アイテム 2	クリア条件
↓	↓	↓	↓	↓	↓	↓	↓
1	0	0	1	0	0	1	1

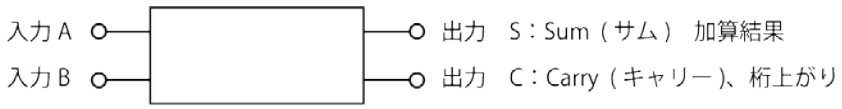
この状態で、現在クリア条件を満たしているか知りたい場合 bit 番号 0 の状態を知りたい。そのため 1～7 を 0 にして bit 番号 0 だけ取り出すには下記のようにすればいい。

```
        1001  0011
AND     0000  0001   ← 抜き出したい bit だけ 1 にする
        0000  0001   → これで bit 番号 0 の情報が抜き出せた
              ↑
ここは必ず “0” になる
```

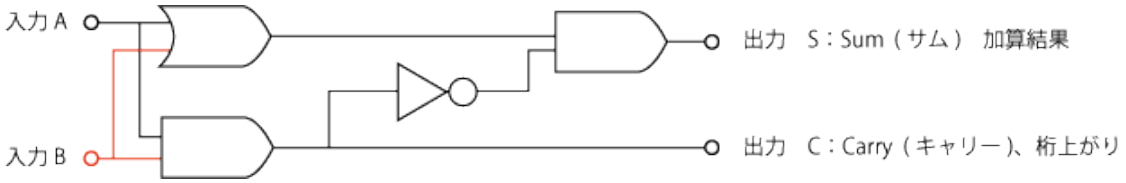
3.9. 半加算回路

今まで学習してきた論理演算を使って、コンピュータで計算することができる回路(演算回路)を考えてみる。
演算回路の中で最も基本的なものが 「半加算回路」 になります。

半加算回路 ： 2 進数 1 桁だけ計算、桁上がりを実装する。

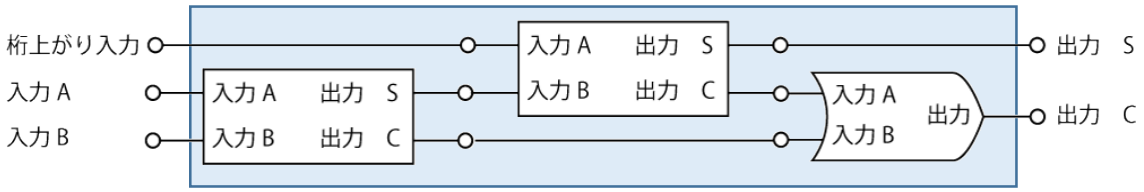


これを論理回路で実装したものが下記のようになる

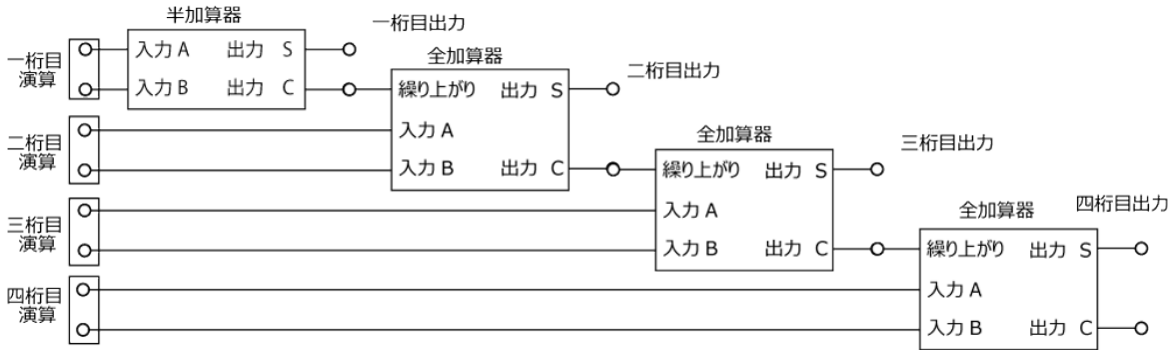


3.10. 全加算回路

下の桁の繰り上がりも考慮して 2 進数 1 桁の加算を行うことができる回路になります。



全加算機と半加算器を組み合わせた演算回路



3.11. メモリ

SRAM(エスラム) と DRAM(ディーラム)の違い。

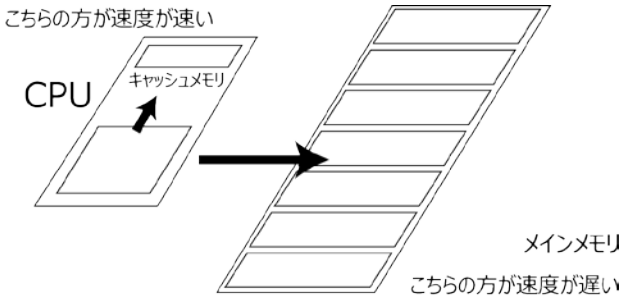
SRAM：高価だが早い、フィリップフロップにより構成されている。
DRAM：低価格だが、リフレッシュが必要であり、比較的遅い。

※キーワード： フリップフロップ

3.12. キャッシュ・メモリ

メインメモリには主に DRAM が利用されるが速度が遅い。そのため何回も同じ命令/データを利用する際に、より高速な「キャッシュメモリ」と呼ばれるメモリを配置し、全体の高速化を図ります。ライトバック、ライトスルーなどいくつか構成方法があります。

また、アクセスする際、キャッシュに保管されたデータが利用できる率を 「キャッシュ率」 と言います。



3.13. 実行メモリアクセス時間

教科書 P110

CPU がメモリにアクセスする際、ヒット率が 80% → メインメモリへのアクセスが 100% - 80% = 20% 発生する

	アクセス時間	アクセス頻度	実アクセス時間
キャッシュメモリ	10 ns	0.8	10 × 0.8 = 8 (ns)
主記憶装置	90 ns	1.0 – 0.8 = 0.2	9
			26 (ns)

3.14. ※入出力装置は割愛します

3.15. 入出力インターフェイス

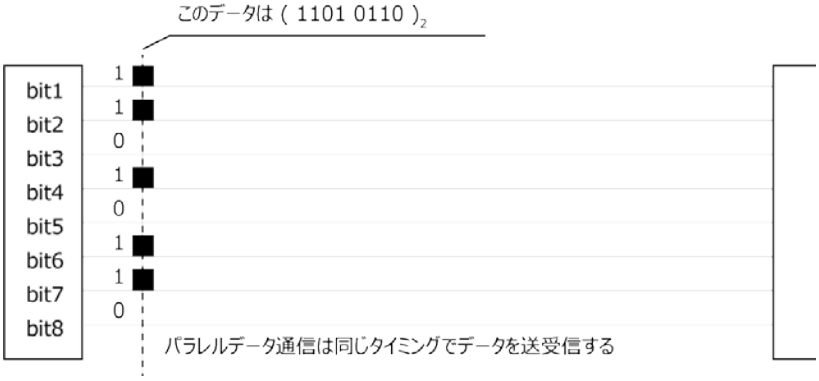
3.15.1. シリアルインターフェイス

基本的に 1 本のデータ線でデータの送受信を行う。差動回路を利用することで高速化が容易にできるため、現在一般的な方式となっている。

シリアル通信の波形

3.15.2. パラレルインターフェース

データのビット数だけデータ線を用意し、複数のデータ線を同時に利用して送受信する方法。一見こちらの方が高速化できそうだが、高速化すればするほどクロストークや、タイミングの問題が発生するため現在ではほとんど利用されていない。



3.15.3. USB

最近よく出題される。もともとマウスやキーボードなど低速度の機器を、ホットプラグイン(電源を入れたまま接続、切断できる)などの機能により手軽に接続するために策定された。

しかし現在では、通信速度の向上も話題だが。電力送信の規格も追加され（USB-PD）注目されている。

1.4.3.1. USB 参照。 接続はバス型

3.15.4. IEEE1394(FireWire)

策定時、USB よりも高速だったが、現在では同等です。ポートの利用料が必要だったため余り広がらなかったが、主に Sony や Apple で利用されている。

機器間の接続はデイジーチェーン

4. 応用数学

情報処理関連で利用する考え方になります。公式や表記が少し難しいと思いますが、実際の状況を客観的に把握するために利用できるものが多く、知っていると得です。

4.1. 階乗

１から４を全て掛けた結果 ＝ 1×２×３×４ ＝ 24

これが １から 1,000 まで掛けた結果・・・ 1×２×３×４× … 1,000 回書くのか？

面倒なので １から 1,000 まで掛けた数字を 1,000！ と表現したものが **階乗** になります。

ただ 0! ＝ 1

4.2. 順列

限られた要素の中から、順番を考慮しながら取り出す際のパターン数を計算することができます。

相異なる n 個の要素から構成される集合から、重複しないように r 個取り出して 1 列に並べる。

6 個のアルファベットを使って **4 文字**の文字を作成することを考えます。

4 文字目を B とした場合取り出された文字列は

A G J B であって ≠ G B A J ではない。

これを順列 (Permutation : パーミュテーション) という

数式では下記の通りになる

6 通り × 5 通り × 4 通り × 3 通り

$$ {}_nPr = n\times(n-1)\times\cdots\times(n-r+1) $$

$$ =\frac{n!}{(n-r)!} \quad \frac{6!}{(6-4)!}=\frac{6!}{2!}=\frac{6\times5\times4\times3\times\cancel{2\times1}}{\cancel{2\times1}} $$

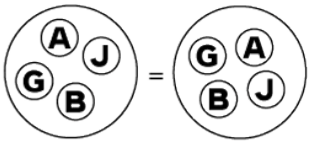
4.3. 組み合わせ

限られた要素の中から、順番を考慮せずに取り出す際のパターン数を計算することができます。

相異なる n 個の要素から構成される集合から、重複しないように r 個取り出して 1 列に並べる。

6 個のアルファベットを使って **4 文字**の文字を作成することを考えます。

順番を考えない場合 4 文字目を B とすると
取り出された文字のかたまりは同じと考えます。



重複分は下記のように、取り出す個数と、
要素数が同じ組合せを考えればいい。

A G J B

G J B A

つまり $_rP_r = r!$

これを組合せ (Combination : コンビネーション) という

数式では下記の通りになる

$$_nC_r = \frac{_nP_r}{r!} = \frac{_nP_r}{r!} = \frac{n!}{(n-r)! \times r!}$$

$$\Rightarrow \frac{6!}{(6-4)! \times 4!} = \frac{6!}{2! \times 4!} = \frac{\overset{3}{\cancel{6}} \times 5 \times \cancel{4 \times 3 \times 2 \times 1}}{\cancel{2} \times 1 \times \cancel{4 \times 3 \times 2 \times 1}}$$