

計算機構成論

Lecture 4

コンピュータでの計算の実行方法

2023年度春学期

情報理工学部 Rクラス担当

越智裕之

内容

- MIPSについて
- プログラム内蔵方式での実行の仕組み
 - 構成要素：ALU、メモリ、レジスタ、PC
 - 実行の様子：C言語→アセンブリ言語→機械語

プログラム内蔵方式の概要をきちんと理解：
特に命令実行時に主記憶の状態がどうなっているか

- 教材：教科書2.3節までを本スライドで補足

命令と命令セット

命令 (instruction) : コンピュータのハードウェア（より具体的にはプロセッサ）への指示語

: 各プロセッサが理解できる命令の集合

- 命令を実行順に並べるとプログラムができる
- プログラムを作るためには様々な種類の命令が必要
 - 整数演算、浮動小数点演算、メモリの読み書き、条件分岐、例外処理、...
- プロセッサが実行可能な命令の種類によって、プロセッサの個性が生じる(コスト、性能、消費電力、アプリによる得意不得意、...)
- **命令セットはプロセッサによって異なる**
 - Intel の x86 や x64
 - ARM の v7 や v8
 - MIPS Technologies の MIPS

MIPSについて

この授業では、MIPS Technologies社の MIPS の命令セットを題材として説明する

MIPS: **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages

- Simple is the bestの発想で設計された

参考：MIPS という略語はかつて（クロック周波数が数MHz程度だった時代）、計算機の性能の単位（Million Instructions Per Second: 毎秒何百万個の命令を実行できるか）としても用いられていた



内容

- MIPSについて
- プログラム内蔵方式での実行の仕組み
 - 構成要素：ALU、メモリ、レジスタ、PC
 - 実行の様子：C言語→アセンブリ言語→機械語

プログラム内蔵方式の概要をきちんと理解：
特に命令実行時に主記憶の状態がどうなっているか

- 教材：教科書2.3節までを本スライドで補足

命令実行に必要なハードウェアの構成要素

- ALU: 算術論理演算ユニット  の略
 - 算術演算と論理演算をする演算器
- メモリ: 実行するプログラムやデータを格納
 - 容量の大きい主記憶
 - 容量が小さいが高速なキャッシュメモリ
- レジスタ: 高速なプロセッサ内のメモリ
 - MIPSではALUの入出力となる（他のほとんどのアーキテクチャでも）
- PC:  の略
 - 「現在実行中の命令」が格納されているアドレスを保持

P28

註：この文脈で PC = Personal Computer（パソコン）だと勘違いしたら一発退場

ワードとバイト

• Word: コンピュータで扱うデータ量の単位

レジスタやALUで一度に扱うデータ量が1ワード

MIPSでは 1ワード = 32ビット

• Byte = ビット (普通)

バイト・アドレッシング方式

MIPSでは

バイト単位でメモリに格納 (アドレス付け)

ワード単位でメモリから読み書き

参考 : C言語で

右の型の変数は (普通) 何バイト?

char

short int

int

float

double



バイト



バイト



バイト



バイト

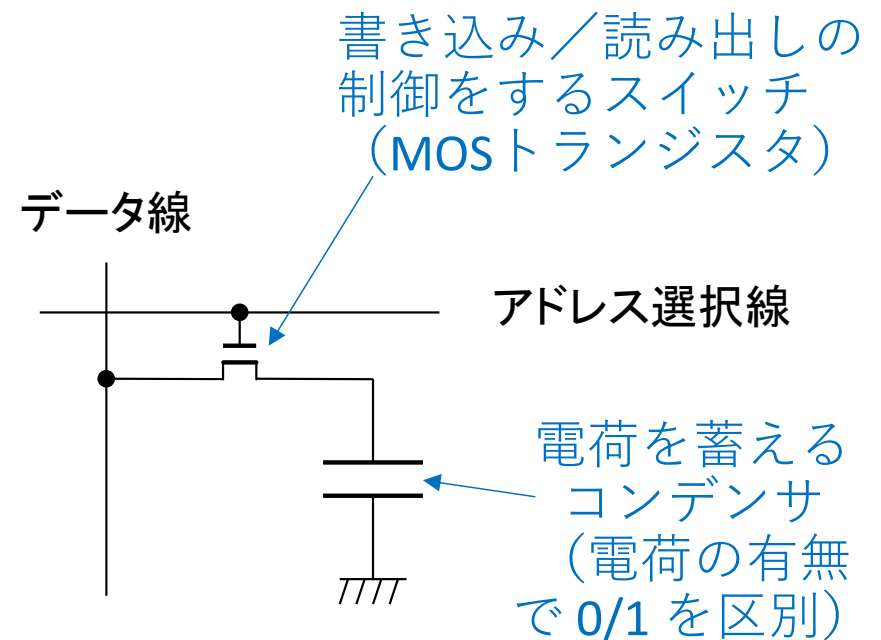


バイト

主記憶

- 主記憶は，順番に が振られた一列に並んだ記憶セル
- 通常，各々のセルは1バイト = ビットのビット列を記憶
- DRAM** が使用される.

番地	← 8bit →
111...111	00101011
111...110	10110101
⋮	⋮
000...011	01010000
000...010	00010010
000...001	10101111
000...000	11111101



DRAMの1ビット分の記憶素子の構成

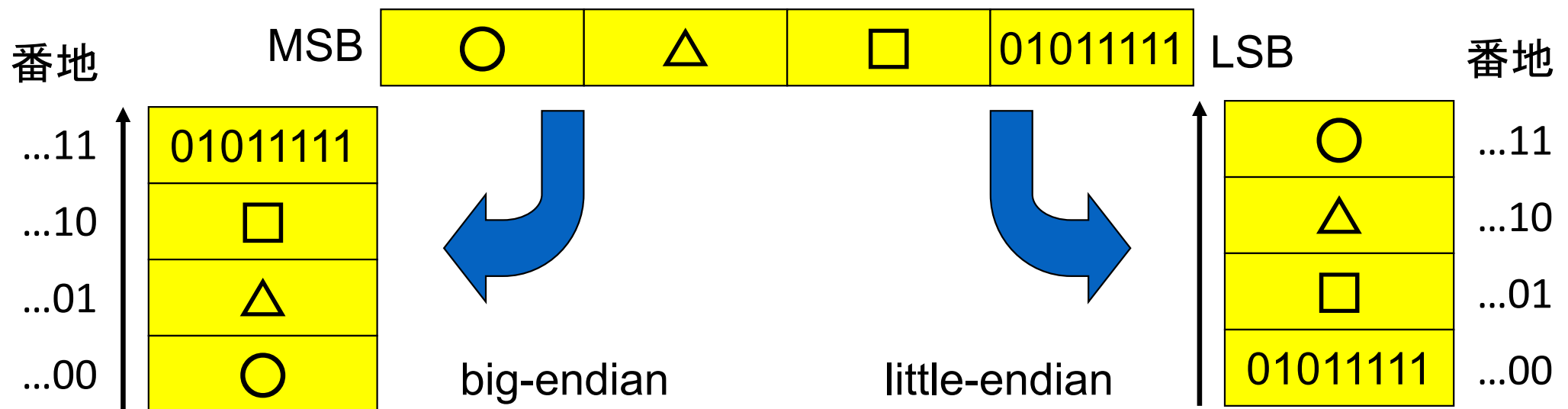
※ この授業でメモリを図示する時は、小さいアドレスを下側、大きいアドレスを上側に描く。

主記憶へのデータの格納方法

プロセッサは8ビット幅以上のデータを **big-endian** か **little-endian** のいずれかの形式で主記憶に格納

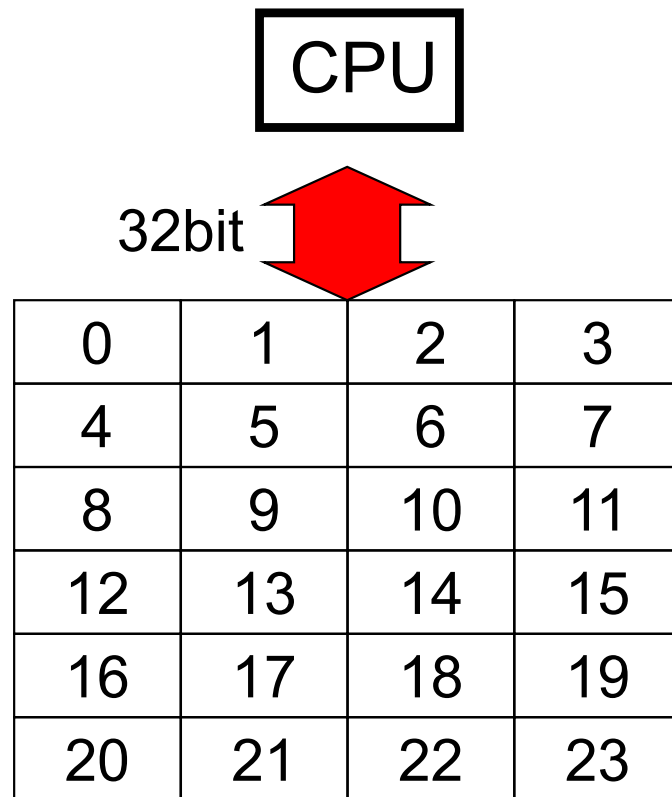
- big-endian: MSB側から8bitずつ順番に格納 (MIPS)
- little-endian: LSB側から8bitずつ順番に格納 (Pentium)

ミニクイズ : MSB、LSBは何の略？



主記憶へのワードの格納

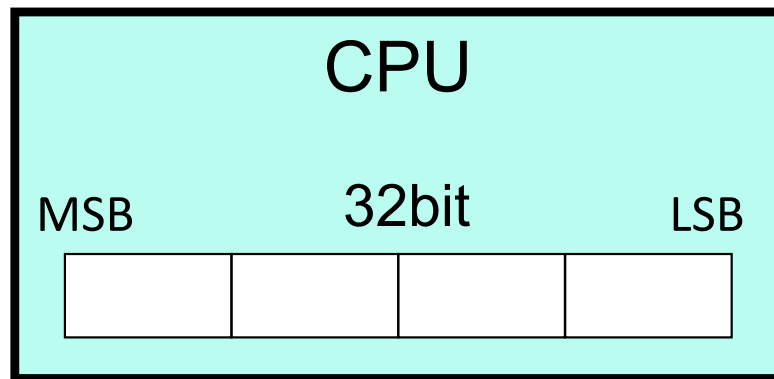
整列化制約: MIPS では、主記憶アクセスの高速化のため、ワードは4の倍数の番地が先頭になるように配置しなければならない。



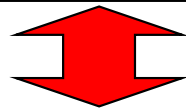
質問：int型の変数を番地14から配置した場合、何が問題か？

主記憶へのワードの格納

整列化制約: MIPS では、主記憶アクセスの高速化のため、ワードは4の倍数の番地が先頭になるように配置しなければならない。



質問：int型の変数を番地14から配置した場合、何が問題か？



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23

ミニクイズ

MIPSにおいて、C言語で、`int A[8];`と宣言した。

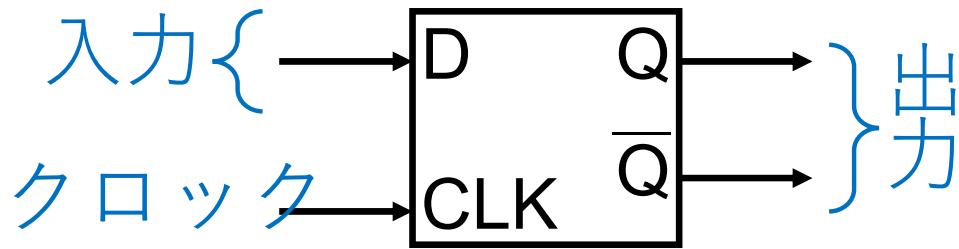
A[0]が格納されているメモリセルの先頭アドレスが0の時、A[7]が格納されているメモリセルの先頭アドレスは何か？

doubleならどうなるか考えよ？

レジスタ

- レジスタは、Dフリップフロップで構成される。
- プロセッサは、主記憶に格納されているデータよりも、レジスタに格納されているデータを**より高速に**読み書きできる。
 - 頻繁に参照されるデータを一時的に記憶する
 - 演算の結果の一時的保存
- プロセッサには限られた数のレジスタしかない。
 - MIPS の例：\$s0, \$s1, ..., \$s7, \$t0, ..., \$t9 等の名前の**32**ビット幅のレジスタを**32**本（1ワード = bit）
 - Intel Pentium の例：EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, ..., 等の名前の**32**ビット幅の一般用レジスタを8本。

Dフリップフロップ

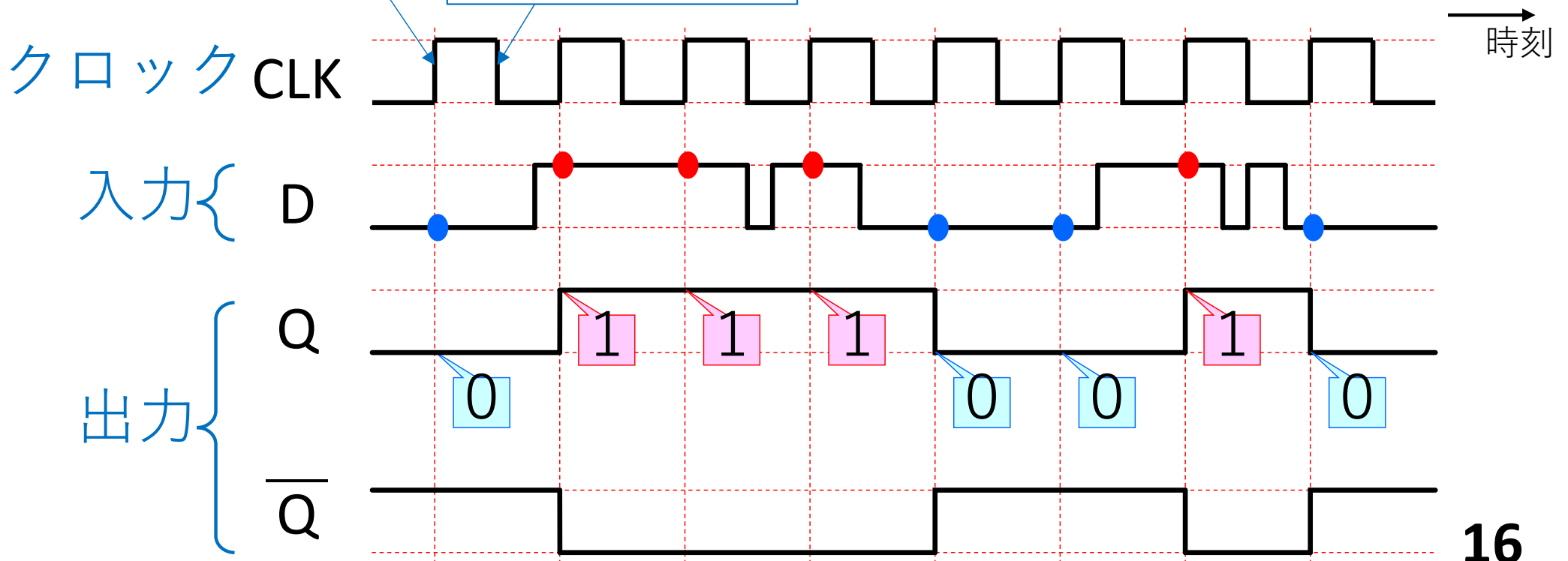


クロックが立ち上がる度に
その時の入力Dの値に従って
出力Qが更新される

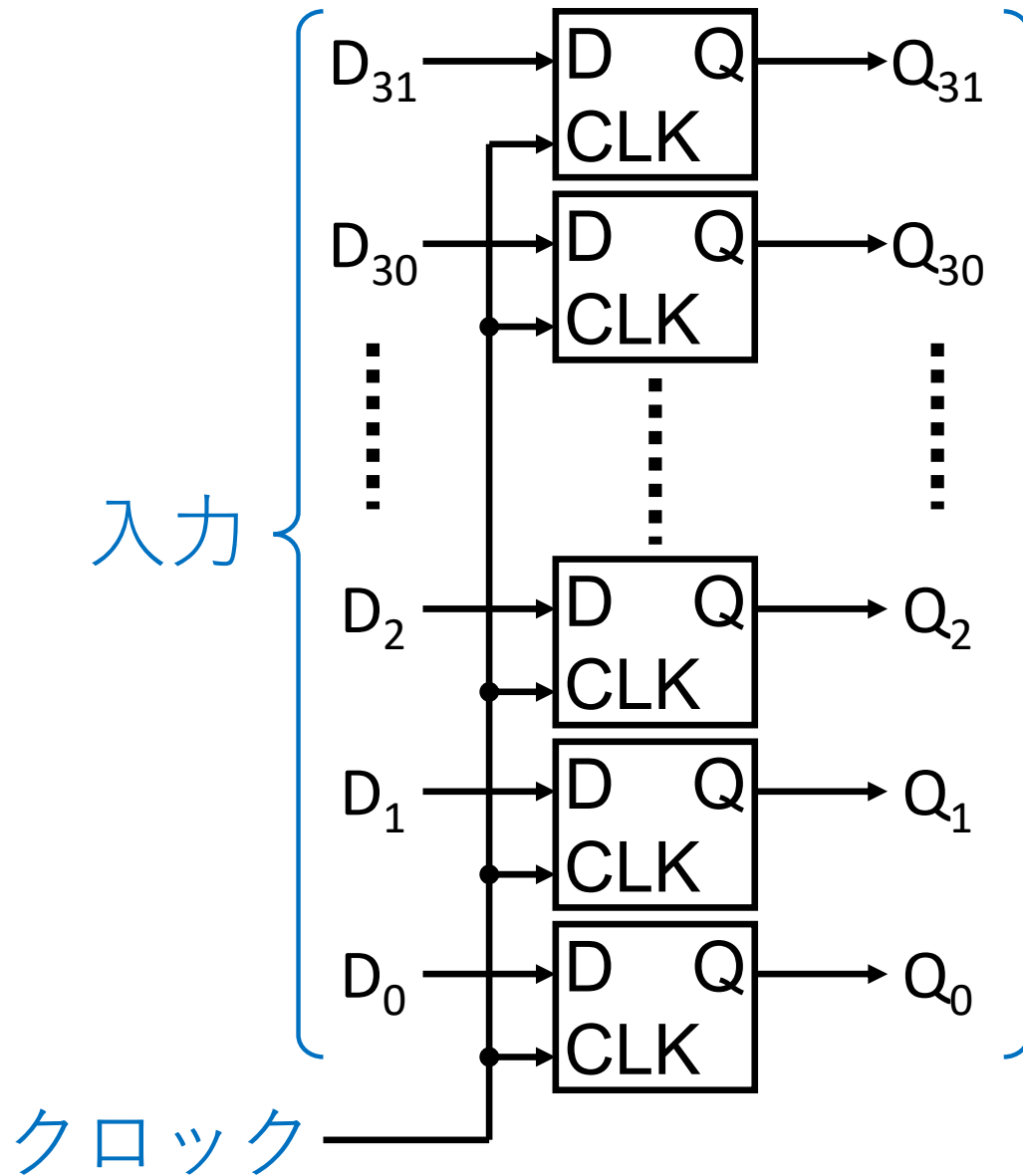
Leading Edge
立ち上がりエッジ

Trailing Edge
立ち下がりエッジ

クロックが立ち上がる時以外は
Dが変化しても出力に影響しない
(リーディングエッジトリガ型)



単純な32bitレジスタ



クロックが立ち上がる度に
その時の入力 $D_{31} \sim D_0$ の値に従って出力 $Q_{31} \sim Q_0$ が更新される

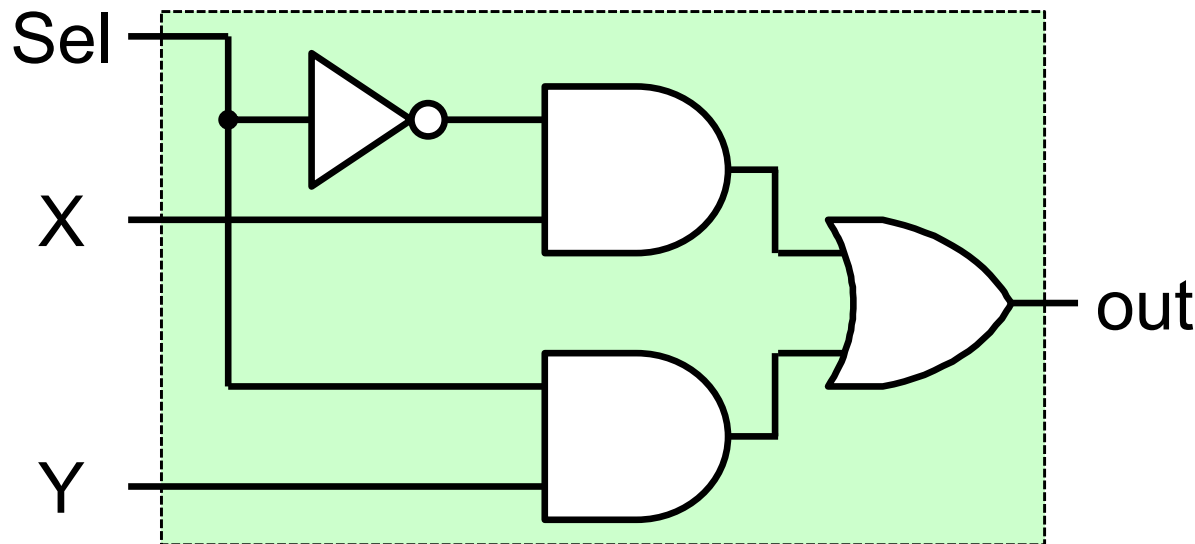
- フリップフロップ32個で32bitのデータを1個保持することができる
- この単純なレジスタは、クロックが立ち上がる度に必ず値が更新されてしまう

2-1 MUX: マルチプレクサ (セレクタ)

脱線

2-1 MUXの論理式: $\text{out} = \overline{(\text{Sel})} (X) + (\text{Sel})(Y)$

Selが0の時X, Selが1の時Yを選ぶ論理



2-1 MUXの回路

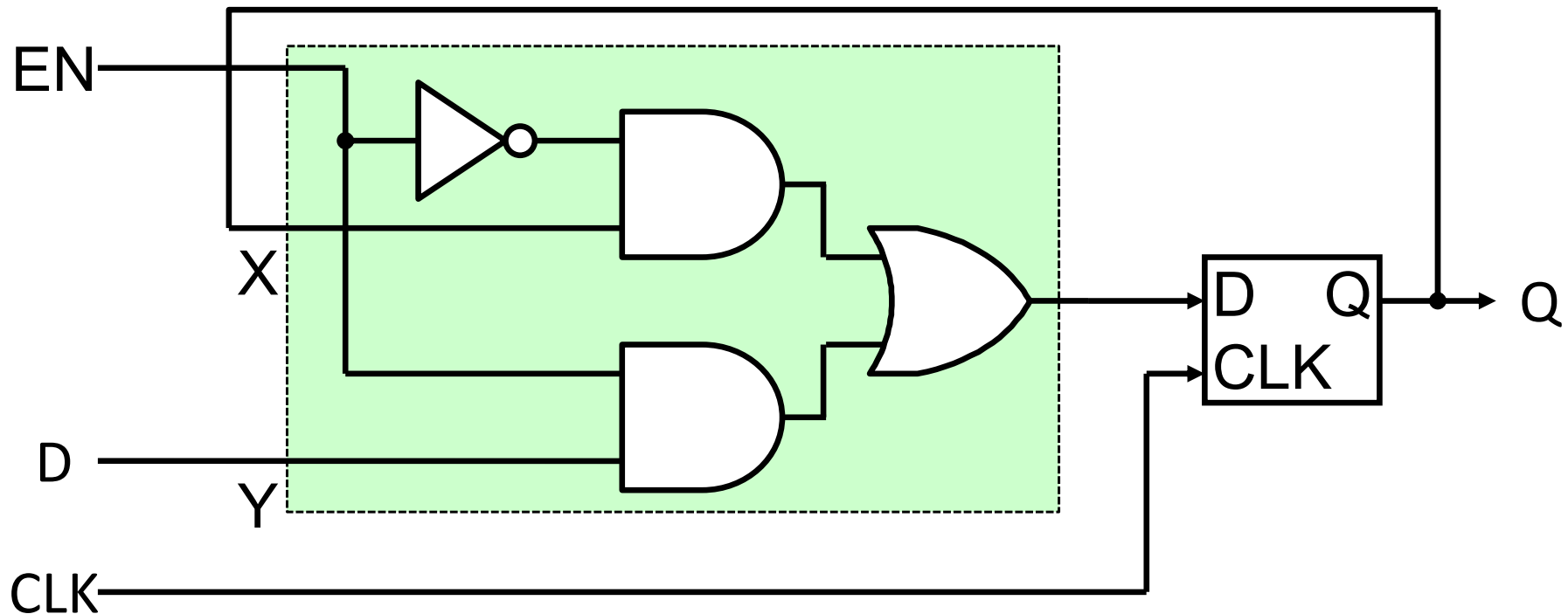
制御信号で AND ゲートを開閉するという動作を考えれば、論理式を作らなくても直接回路を書けるはず

書き込み制御信号付きフリップフロップ

脱線

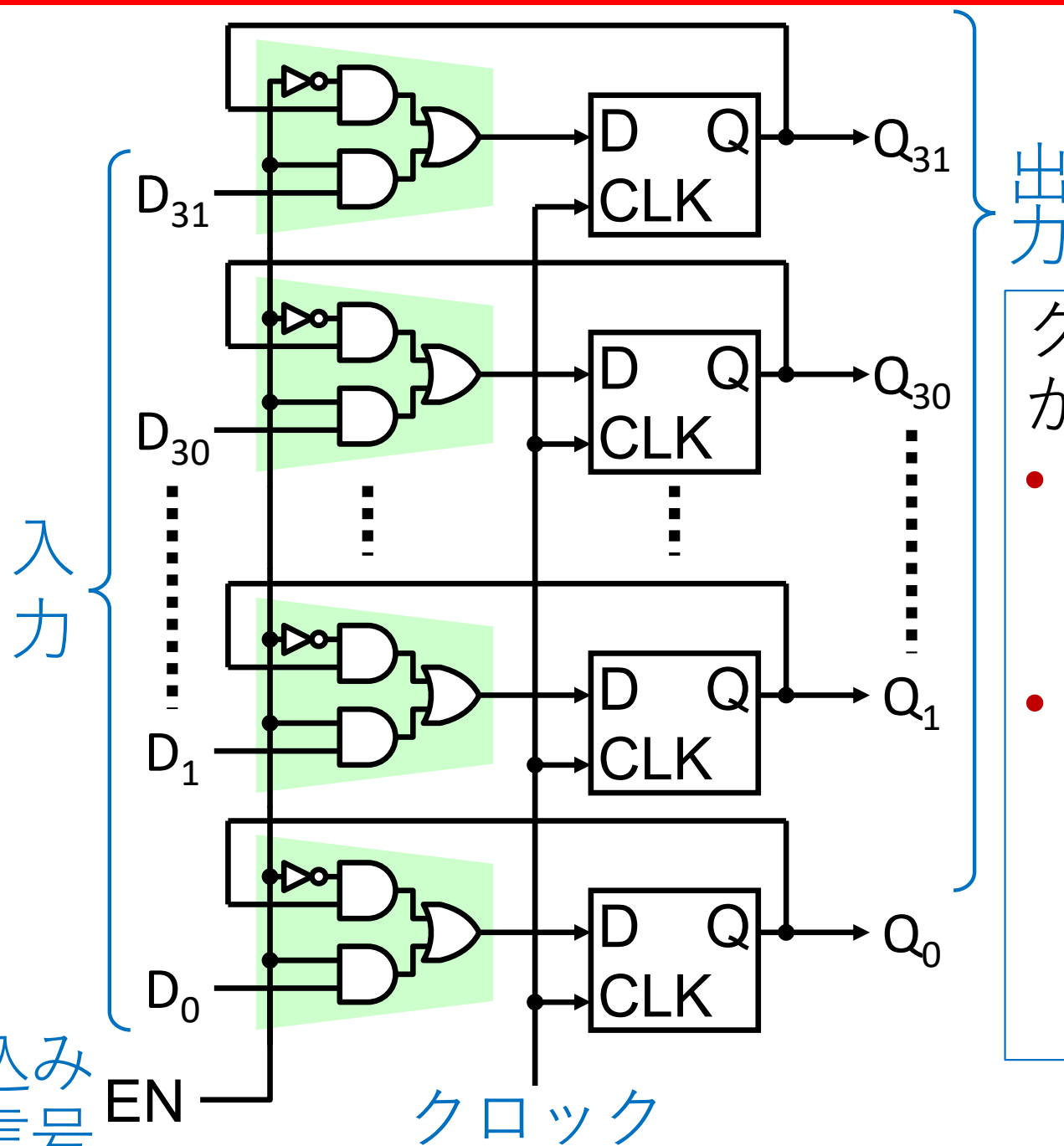
クロック CLK が立ち上がった時、

- **EN=0 ならば、** 現在のフリップフロップの値(Q)がフリップフロップに上書きされる **(値は変化しない)**
- **EN=1 ならば、** 入力Dの値がフリップフロップに書き込まれる **(値が更新される)**



書き込み制御信号付きレジスタ

脱線

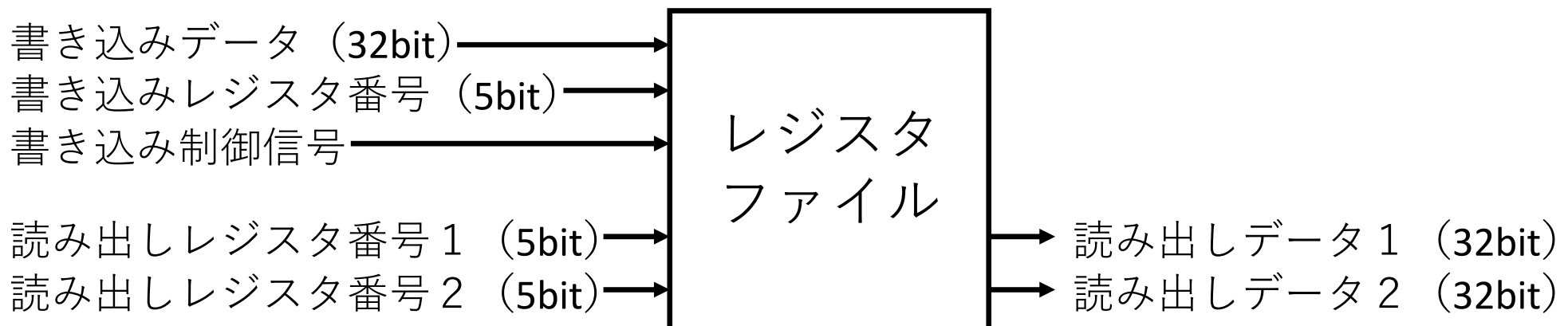


クロックが立ち上がった時、

- $EN=0$ ならば、 $Q_{31} \sim Q_0$ の値は変化しない
- $EN=1$ ならば、その時の入力 $D_{31} \sim D_0$ の値に従って $Q_{31} \sim Q_0$ が更新される

レジスタファイル

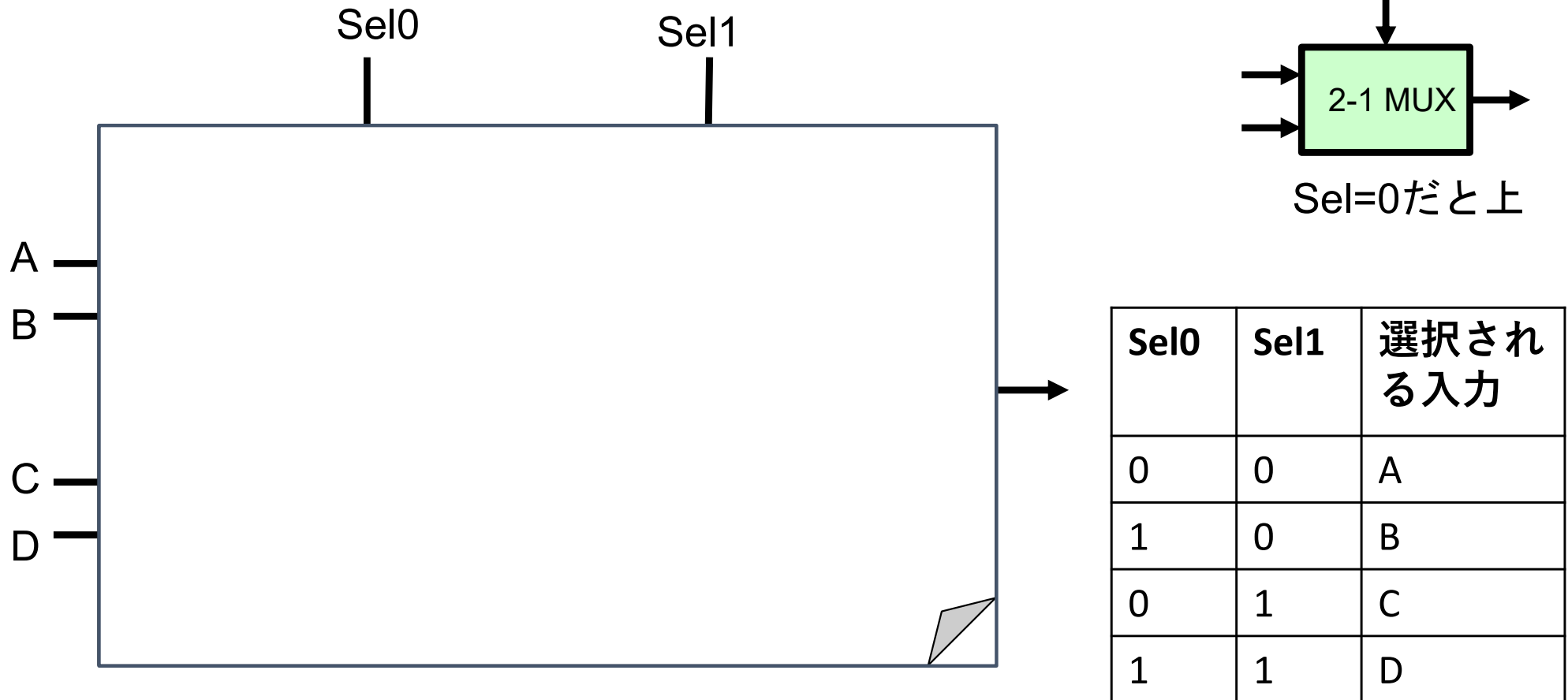
- 複数のレジスタをまとめたもの
- レジスタの番号を指定することによって、個々のレジスタを読み書きできる
- 32bitレジスタを32個有するレジスタファイルの場合
 - フリップフロップ数は $32 \times 32 = 1024$ 個必要
 - レジスタ番号は0番から31番まで
 - 2進数で00000から11111まで（5bit必要）



ミニクイズ：レジスタの選択方法

脱線だけど重要

Sel0、Sel1 の2ビットで、4つのレジスタA,B,C,Dの出力を選択するような回路を、2-1MUXを複数使って構成せよ。



* レジスタ32個だとどうなるか？

* また、その構造が、32ビット分あると想像してください。

必要な 2-1 MUX の数は $32\text{bit} \times 31\text{個} = 992\text{個}$

レジスタの使用方法

- MIPSの場合
 - $\$s0, \$s1, \dots, \$s7$ (レジスタ16番から23番に相当)
 - $\$t0, \$t1, \dots, \$t7$ (レジスタ8番から15番に相当)
 - あとは特殊なレジスタ

レジスタには変数の値を格納:

1. 高速
2. MIPSではメモリの値を直接演算対象とできない

MIPSでは、オペランドはレジスタ3つのみが基本 (教科書p65)
→ 教科書にある設計原則①: 単純性は規則性につながる

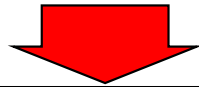
質問: プログラムで使っている変数が8個より多い場合?

Answer: (用語覚えましょう) 教科書p71
レジスタの値 \Rightarrow 主記憶に書き戻す

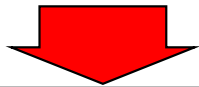
レジスタの数（MIPSでは32）

直感的にはレジスタの数を増やせば速くなるが...

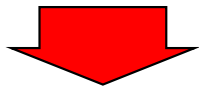
😊レジスタを増やした.



😞回路が大きくなった.



😞配線が長くなって信号の遅延が増えた.



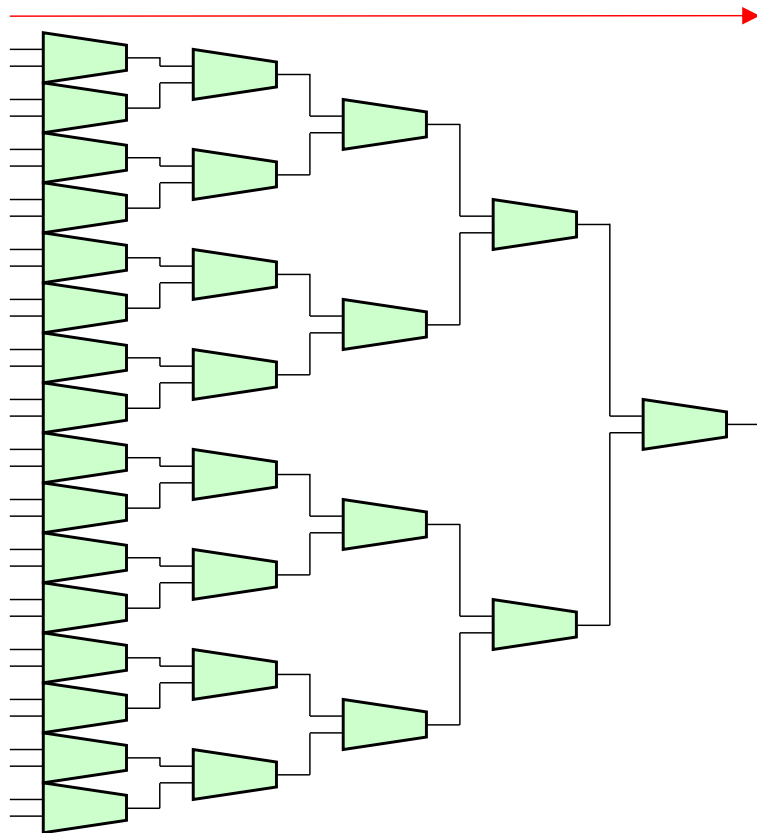
😞クロックの周波数を落とさざるを得なくなった.

教科書にある設計原則②：小さいほど高速

（教科書p67）

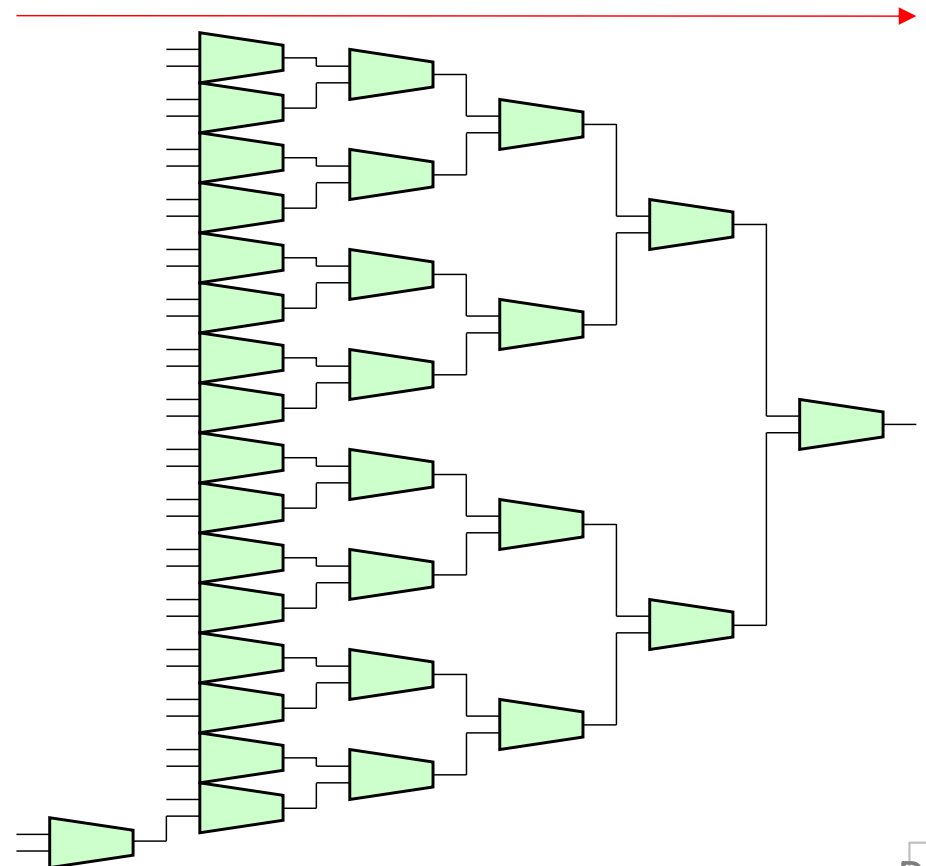
レジスタ数が31→32に増加の場合と32→33へ増加させる時のペナルティは同じか違うか？（演習問題）あと教科書2.3節の最後の自己診断問題も見て下さい。

32-to-1 MUXの場合 通過する2-1 MUXは5段



33-to-1 MUXの場合 通過する2-1 MUXは6段

- 遅延時間が1.2倍に増加
- クロック周波数が0.83倍に低下
- 性能が0.83倍に低下



内容

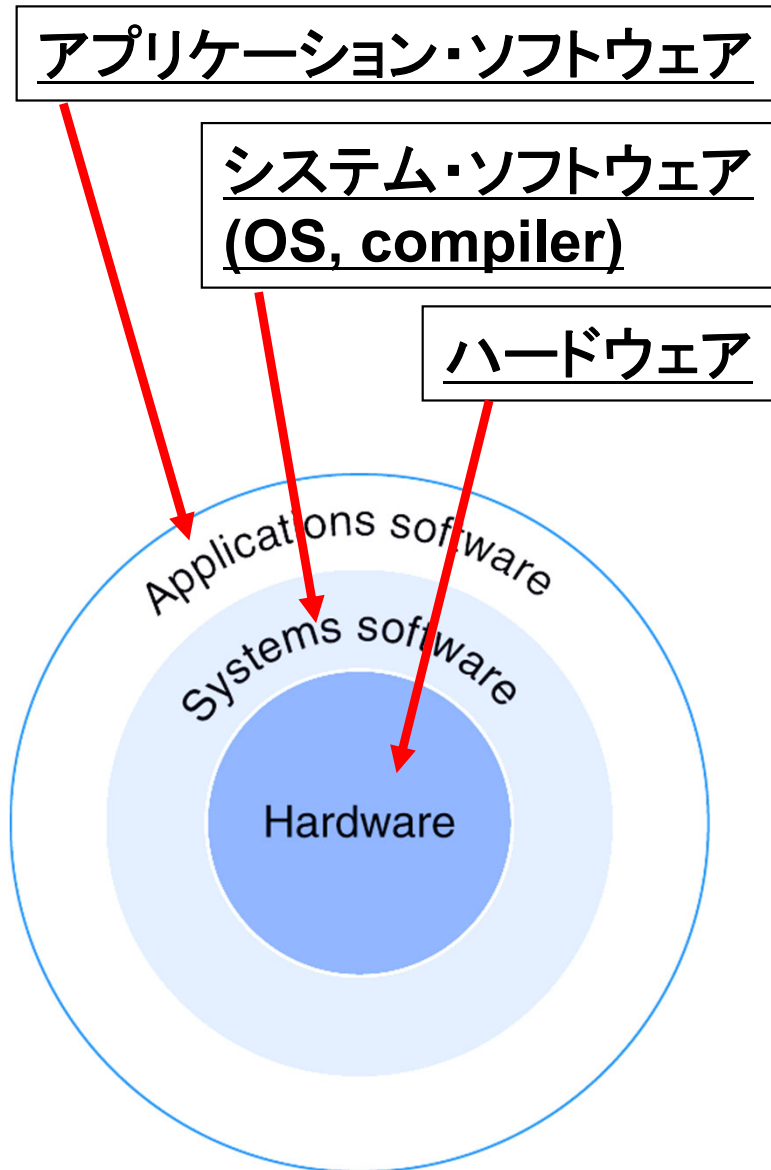
- MIPSについて
- プログラム内蔵方式での実行の仕組み
 - 構成要素：ALU、メモリ、レジスタ、PC
 - 実行の様子：C言語→アセンブリ言語→機械語

プログラム内蔵方式の概要をきちんと理解：
特に命令実行時に主記憶の状態がどうなっているか

- 教材：教科書2.3節までを本スライドで補足

C言語のプログラム実行の裏側

階層(抽象化)



教科書（英語版）図1.3

下線つきの用語は全て理解すべし

高水準言語

$a[i] = b[i] + c;$

↓ コンパイラ

アセンブリ言語

lw \$15, 0(\$2)
add \$16, \$15, \$14
add \$17, \$15, \$13
lw \$18, 0(\$12)
lw \$19, 0(\$17)
add \$20, \$18, \$19
sw \$20, 0(\$16)

↓ アセンブラ

機械語

000000101100000
110100000100010
... (これは適当な値であることを注意)

プログラム内蔵方式とは

- ノイマン型とも呼ばれる

主記憶上に、プログラムの命令列とデータを格納

具体的な実行イメージ

- 「PCのさすアドレスから命令を読んで実行」を繰り返す

PCは

を保持



MIPSでの命令の例 : add \$s1, \$s2, \$s3 ←アセンブリ言語※

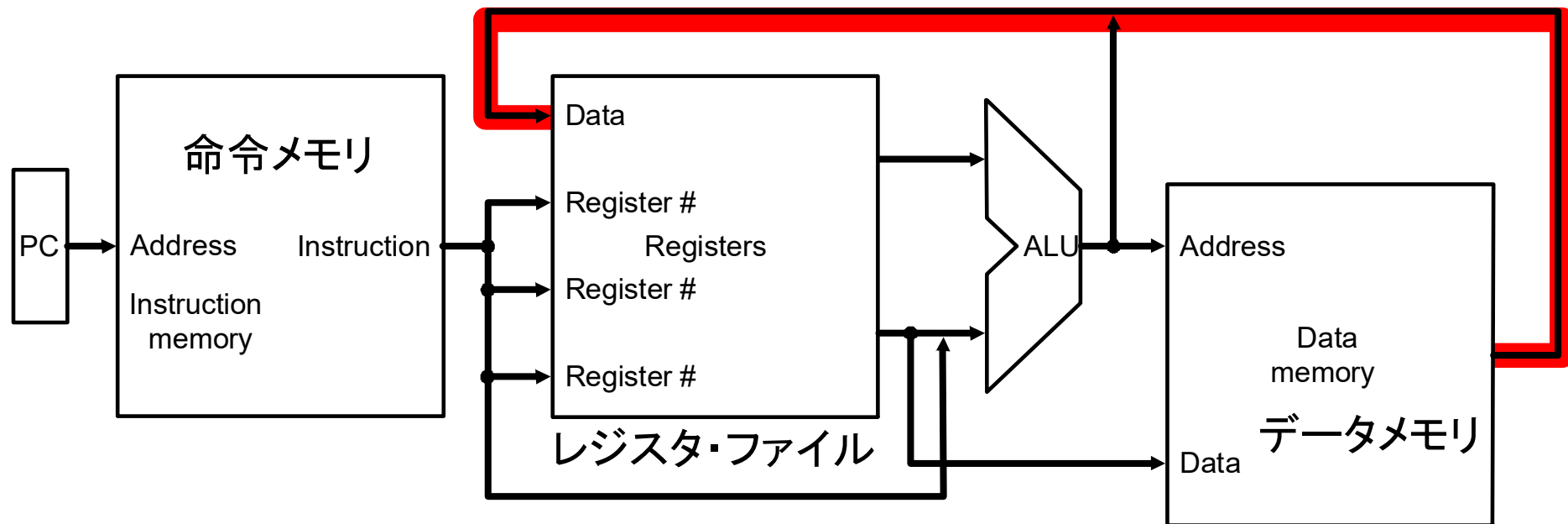
命令

* 原則レジスタ

※ 実際には、0と1の列である機械語を主記憶に格納

データパスと制御の例：ロード命令

ハードウェアの挙動がなんとなくわかればOK

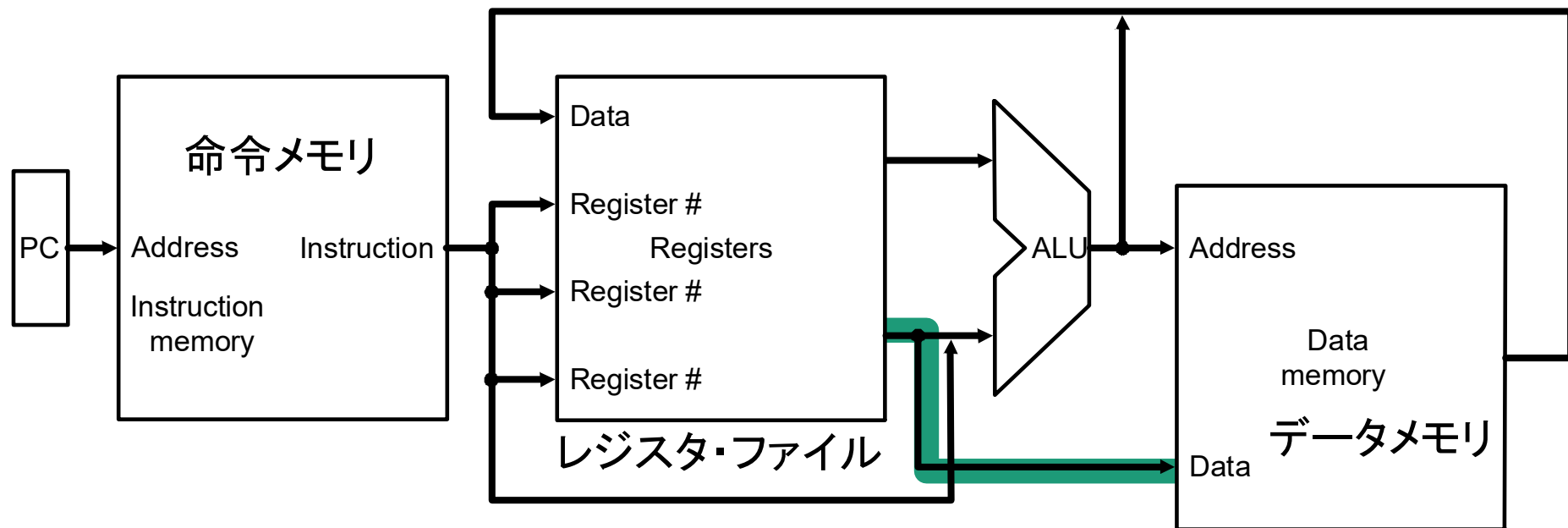


データメモリから読みだされたデータがレジスタ・ファイルに書き込まれる

教科書 図4.1の一部

データパスと制御の例：ストア命令

ハードウェアの挙動がなんとなくわかればOK

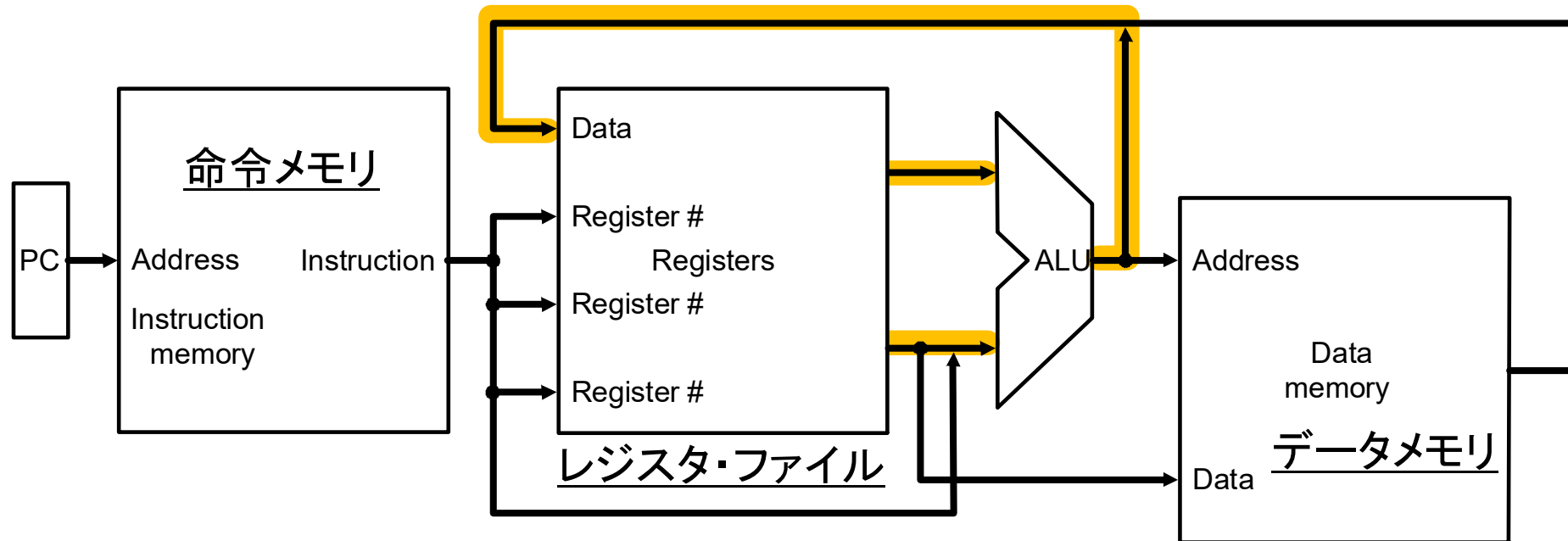


レジスタ・ファイルから読みだされたデータがデータメモリに書き込まれる

教科書 図4.1の一部

データパスと制御の例：R命令

ハードウェアの挙動がなんとなくわかればOK



レジスタ・ファイルから読みだされたデータに対する演算の結果がレジスタ・ファイルに書き込まれる

教科書 図4.1の一部

MIPSの命令:レジスタがオペランドの場合

- MIPS では、0番から31番のレジスタに「**\$△△**」と名前がつけられ、利用目的が想定されている.
- 一般に使用されるレジスタ
 - **\$s0, \$s1, ..., \$s7** (レジスタ16番から23番に相当)
 - **\$t0, \$t1, ..., \$t7** (レジスタ8番から15番に相当)

レジスタオペランドを使用する命令の例:

add	\$t0, \$s1, \$s2	# “\$s1”+ “\$s2” の結果を \$t0 に格納
sub	\$t0, \$t0, \$t1	# “\$t0” – “\$t1” の結果を \$t0 に格納

MIPSの命令: メモリがオペランドの場合

- MIPS では、メモリオペランドを「 $n(\$△△)$ 」と記述し、これは $(“\$△△” + n)$ 番地の内容を意味する。
 - $\$△△$ には、任意のレジスタが指定でき、これを と呼ぶ。
 - n には、符号つき整数が指定でき、これを と呼ぶ。

メモリオペランドを使用する命令の例:

lw	\$t0, 32(\$s0)	# (“\$s0” + 32) 番地の内容を \$t0 に格納
sw	\$t0, -8(\$s0)	# (“\$s0” - 8) 番地に “\$t0” を格納

復習： -8 を16ビットの2進数であらわせ

MIPSの命令:定数の場合 (特殊)



- $A = A + 4$ といった定数との演算は非常に多い
- レジスタしか使えない場合：

```
lw    $t0, 0($s1)
add   $s3, $s3, $t0
```

ミニクイズ：上記の\$s3と\$s1に格納されている値が意味するものは？

そのため、オペランドにレジスタ以外を指定できるものも用意されている

```
addi  $s3, $s3, 4
```

  と呼ばれる

C → アセンブリ言語

$d[3] = d[2] + a;$ をコンパイラでコンパイルすると

(* 変数はすべてint型とする)

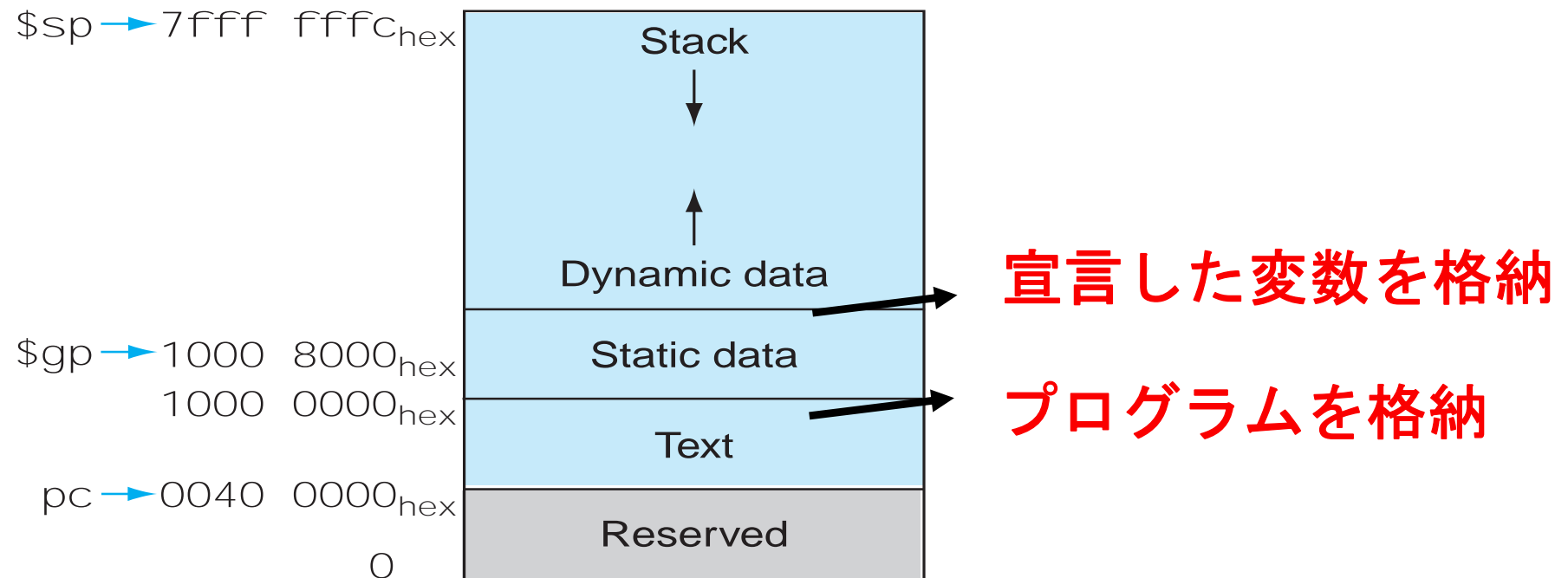
```
lw    $t0, 8($s4)    # d[2] is brought into $t0
lw    $t1, 0($s1)    # a is brought into $t1
add   $t0, $t0, $t1   # the sum is in $t0
sw    $t0, ($s4) # "$t0" is stored into d[3]
```

上記は 言語と呼ばれる、

実際は、 によって変換された0,1の列の が
計算機の主記憶上に格納される

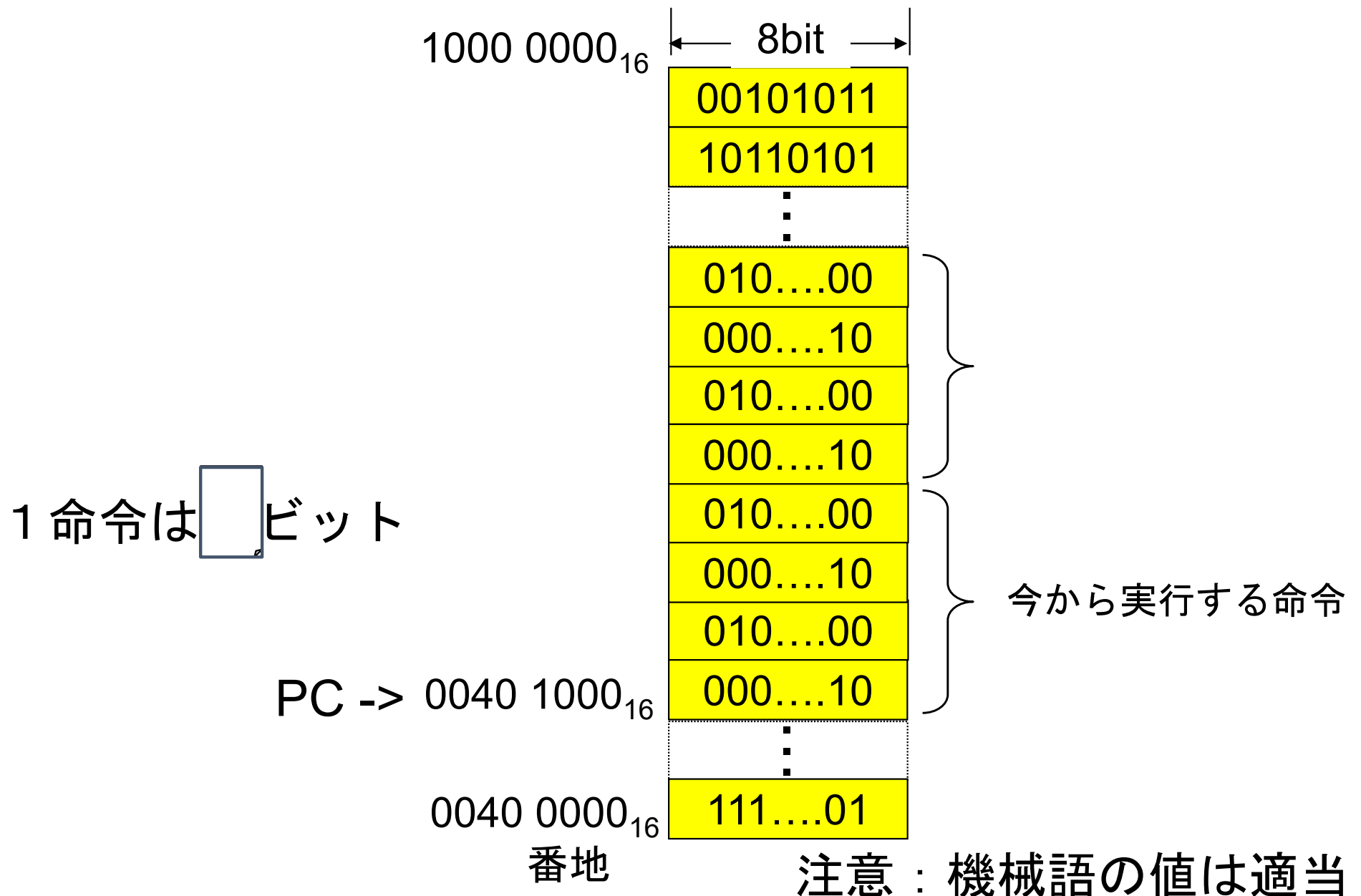
C → アセンブリ言語: 主記憶の様子(全体)

MIPSでの主記憶の様子

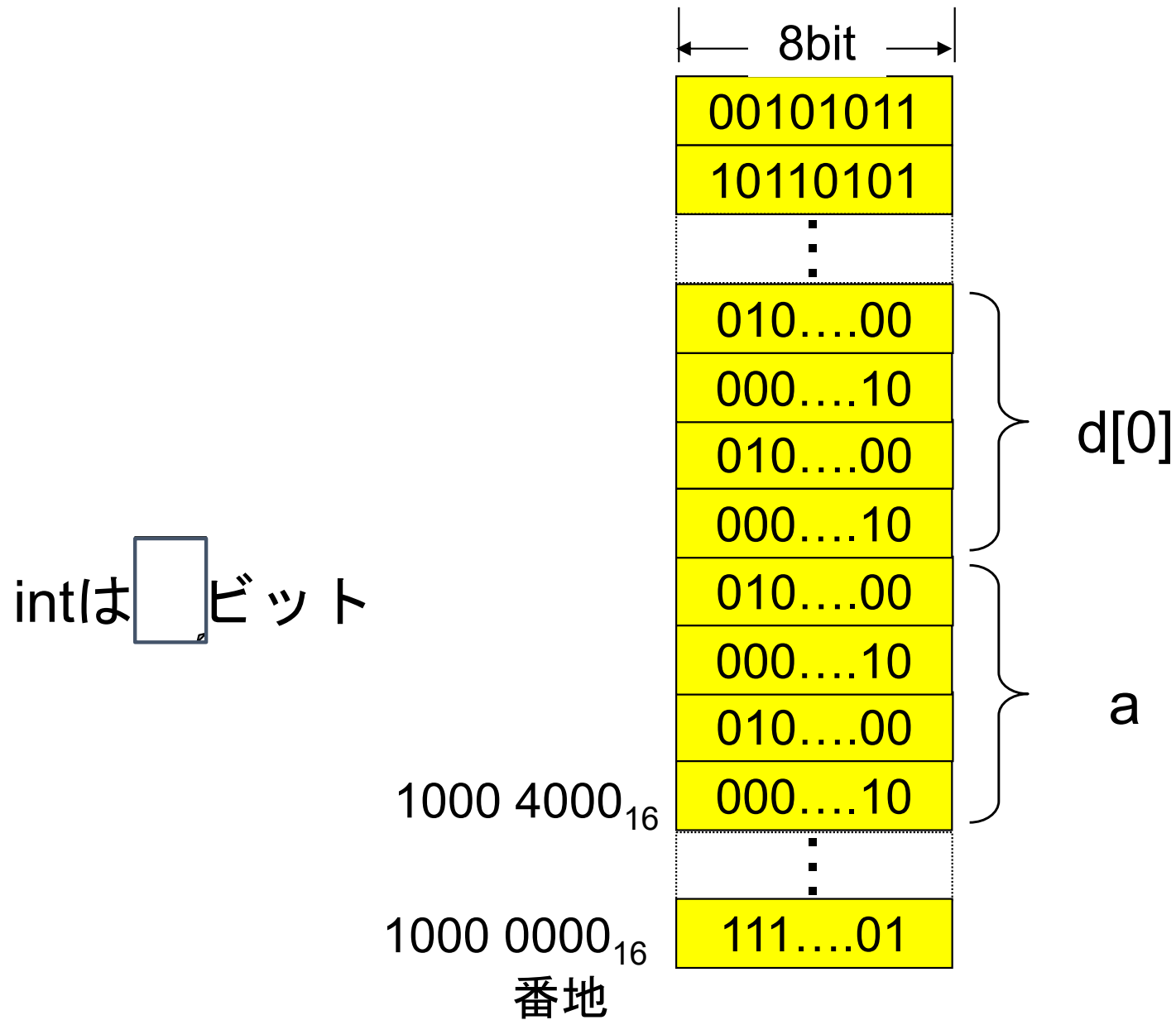


教科書英語版図2.13 (詳細はLec. 6で)

C → アセンブリ言語: 主記憶の様子(命令メモリ)



C → アセンブリ言語: 主記憶の様子(データメモリ)



MIPSの命令のまとめ (Lec. 4の中での)

add	\$t0, \$s1, \$s2	#	<input type="text"/>	の結果を	<input type="text"/>	に格納
-----	------------------	---	----------------------	------	----------------------	-----

sub	\$t0, \$t0, \$t1	#	<input type="text"/>	の結果を	<input type="text"/>	に格納
-----	------------------	---	----------------------	------	----------------------	-----

addi	\$s3, \$s3, 4	#	<input type="text"/>	を \$s3 に格納
------	---------------	---	----------------------	------------

lw	\$t0, 32(\$s0)	#	<input type="text"/>	番地の内容を \$t0 に格納
----	----------------	---	----------------------	-----------------

sw	\$t0, -8(\$s0)	#	<input type="text"/>	番地に "\$t0" を格納
----	----------------	---	----------------------	----------------

演習問題①

①MIPSにおいて、int型の配列A[800]を宣言した。
A[0]が格納されているメモリアドレスが0の時、A[100]
が格納されているメモリセルの先頭アドレスは何か？
さらに、char型、double型ではそれぞれどうか？

②MIPSはbig-endianである。上記の状況で、
A[0]=FF112233 (16進) の時、主記憶のアドレスが0か
ら3の内容、それぞれの8ビットの値を2進数で答えよ。

* 変数のメモリへの配置はコンパイラによって
決定されるが、連続して配置されるとする。

演習問題②

- レジスタ数が32で最適化されているプロセッサのレジスタを33に増やすときに考えられる最も大きい問題は何か？

P25

- アドレス0からint型の変数が格納されているとする。その32bitの値を16進表現にすると、eca8 6420₁₆とする。この時、big-endian および little-endianのそれぞれの形式において、アドレス0から3までのメモリの内容を2進数で表せ。

演習問題③

次のページに示すアセンブリコードに対応する機械語の命令を実行しているとき、主記憶の状態に関して、

- 1つ目の命令が格納されているアドレスが $0040\ 1000_{16}$
- d, aの格納場所が次ページの図のとおりであるとする。（講義と同じ状況）

この時、以下の値は？

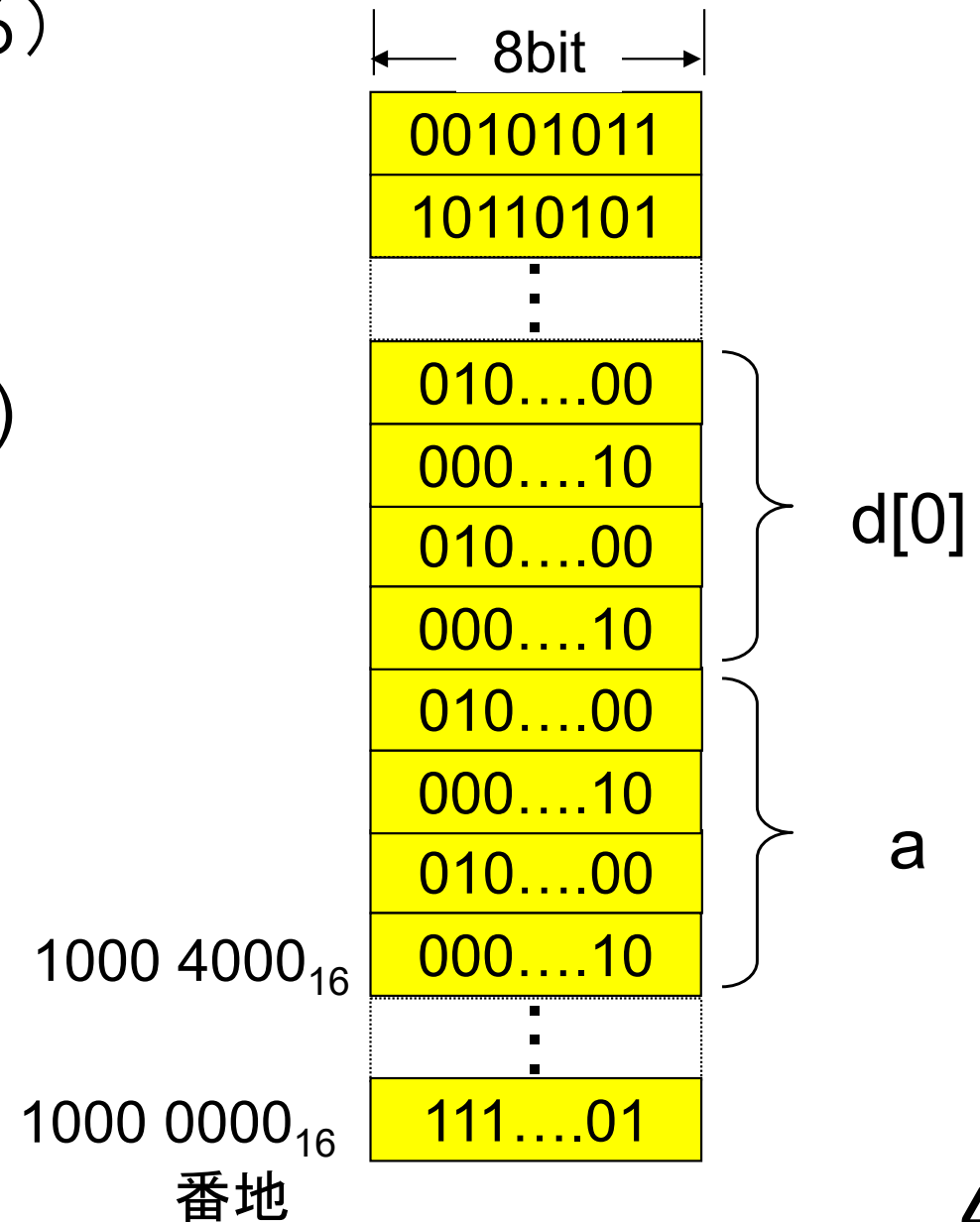
- 4つ目の命令が格納されているメモリの先頭アドレス
- d[5]が格納されているメモリの先頭アドレス
- \$s1と\$s4の値は？（適切な値に設定されるようにコンパイラが適切な命令をこれ以前に挿入しており、その命令によって適切な値が設定されているとする。）
- 最後に、このアセンブリ言語で元のCのプログラムと正しく対応することを説明せよ（＝4つの命令はそれぞれ何をしているかを説明せよ）

演習問題③のための図

$d[3] = d[2] + a;$ をコンパイラでコンパイルすると

(* 変数はすべてint型とする)

```
lw    $t0, 8($s4)
lw    $t1, 0($s1)
add   $t0, $t0, $t1
sw    $t0, 12($s4)
```



Lec. 4での要チェック用語集

MIPS

CISC

RISC

ワード

バイト

オペランド

レジスタ

プログラム内蔵方式

PC

ALU

ベースレジスタ

オフセット

意味と何の略かぐらいはチェックすべし