命令と命令表現

（教科書3.1節～3.4節）

プロセッサの命令と命令セット

- 命令：プロセッサへの指示（プロセッサが実行可能な処理）

加算命令　論理演算命令　減算命令　分岐命令

- 命令セット：プロセッサが実行可能な命令の集合（プログラマから見えるプロセッサの論理仕様）

プロセッサA	プロセッサB	プロセッサC
加算命令 分岐命令	加算命令 減算命令	加算命令 減算命令 論理演算命令 分岐命令

命令セットに含まれない命令は直接実行できない！

プログラム(命令シーケンス)の実行

高水準プログラミング言語
(high-level programming language)

```
if (y == 1)
   c = a + b;
else
   c = a - b;
```

命令セット

コンパイラ
(compiler)

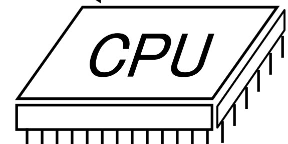
```
Target:
   lw $4, 0($1)
   lw $5, 4($1)
   add $2, $4, $5
```

アセンブリ言語

機械語
(CPUへの命令)

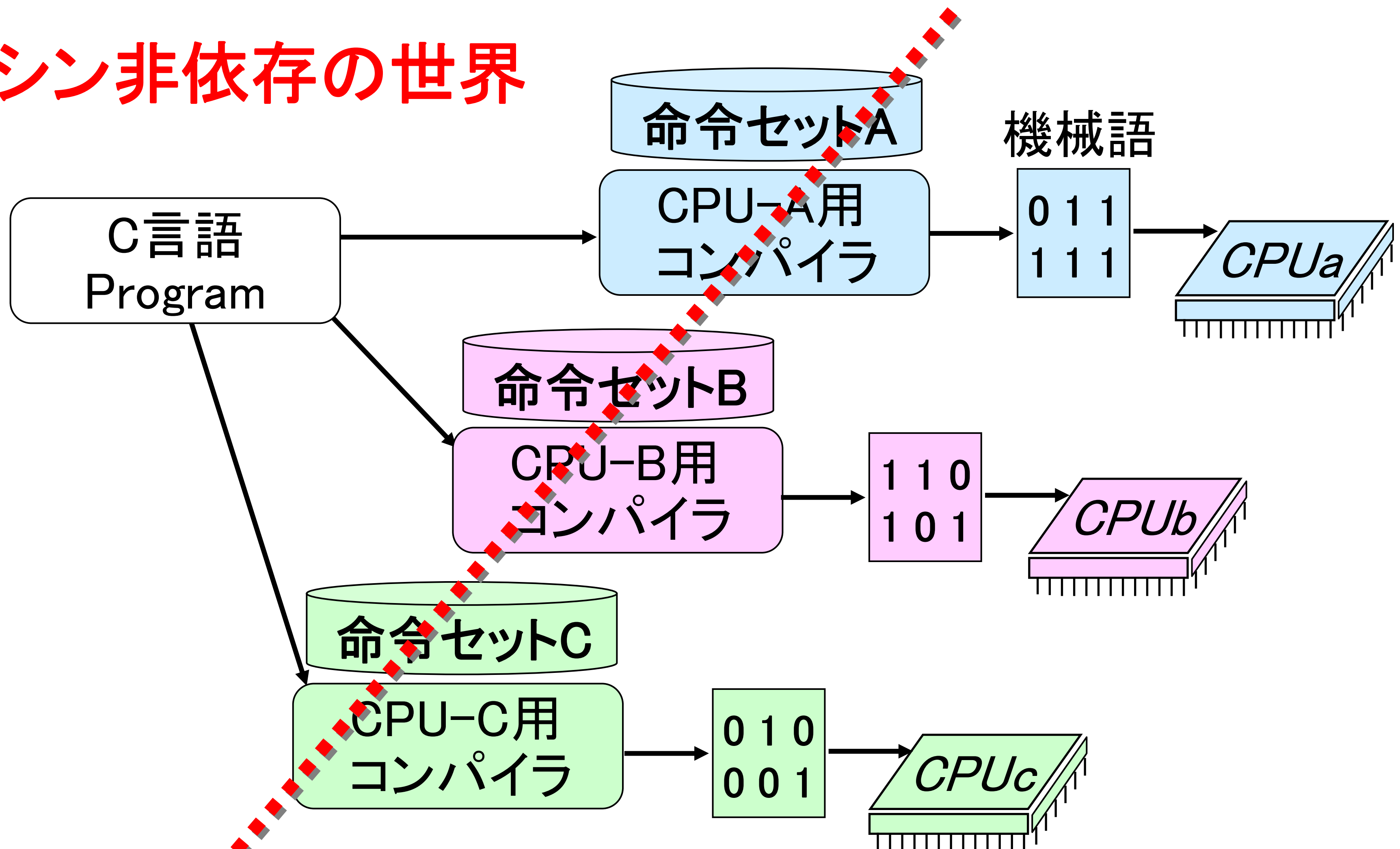
```
1001010010101000
0000010110111000
1110011110100110
```

2進表現



「命令シーケンス」としてプログラムを表現

プログラムとCPUのインタフェース



MIPSとその命令セット

- この授業では MIPS の命令セットを例にする.
  - 基本の考え方はどのプロセッサでもあまり変わらない.
  - MIPS は PlayStation で使用されているプロセッサ.

MIPSの命令セット（R2000/3000）:

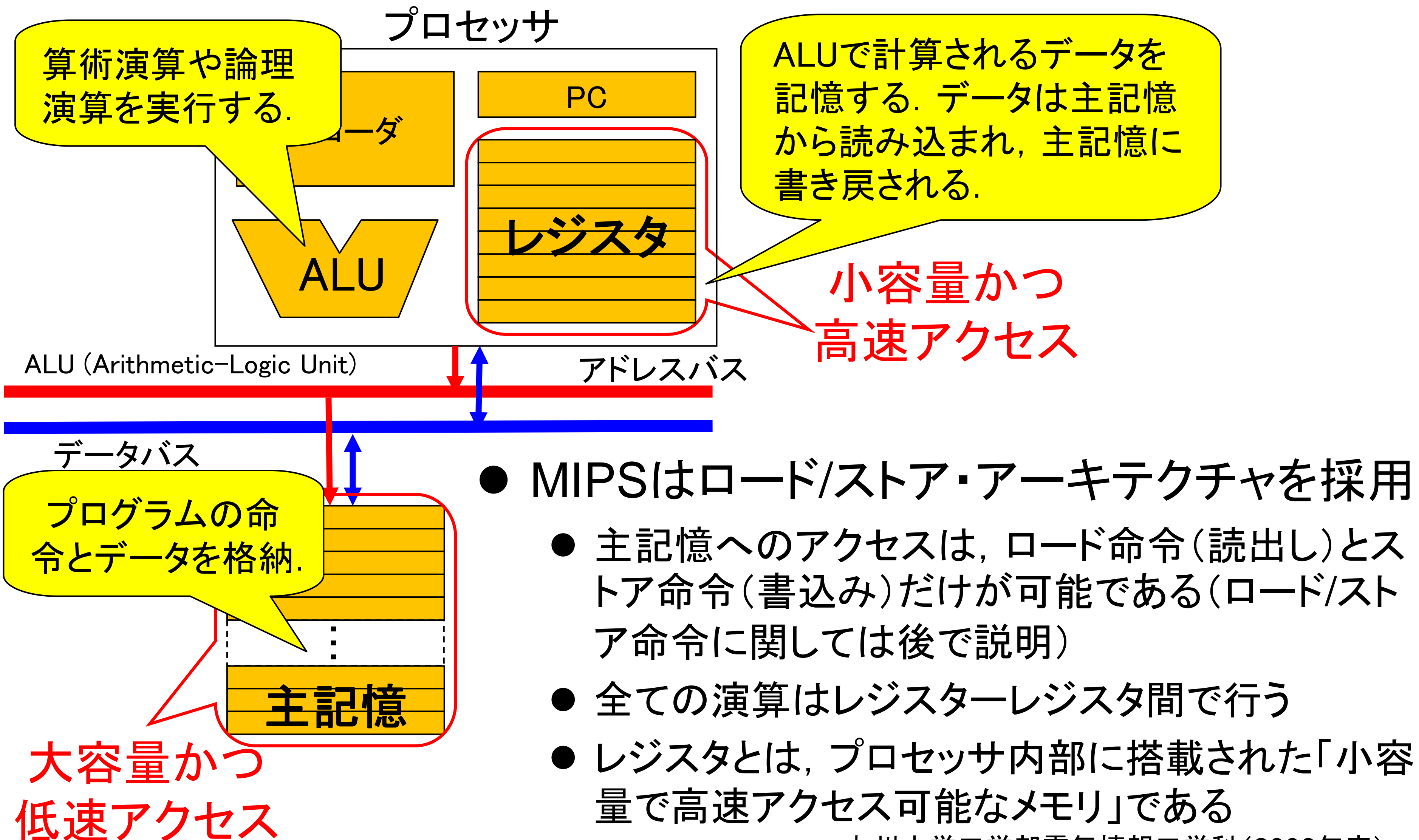
転送命令	LB, BLU, LH, LHU, LUI, LW, LWL, LWR, SB, SH, SW, SWL, SWR
算術演算命令	ADD, ADDI, ADDIU, ADDU, DIV, DIVU, MULT, MULTU, SLT, SLTI, SLTIU, SLTU, SUB, SUBU
論理演算命令	AND, ANDI, NOR, OR, ORI, SLL, SLLV, SRA, SRAV, SRL, SRLV, XOR, XORI
分岐命令	BEQ, BGEZ, BGEZAL, BGTZ, BLEZ, BLTZ, BLTZAL, BNE, J, JAL, JALR, JR
その他の命令	BCzF, BCzT, BREAK, CFCz, COPz, CTCz, LWCz, MFC0, MFCz, MFHI, MFLO, MTC0, MTCz, MTHI, MTL0, RFE, SWCz, SYSCALL, TLBP, TLBR, TLBWI, TLBWR

MIPSの命令（一部）

命令区分	命令	例	意味
算術演算	add	add $s1, $s2, $s3	$s1 = $s2 + $s3
	Subtract	sub $s1, $s2, $s3	$s1 = $s2 − $s3
データ転送	load word	lw $s1, 100($s2)	$s1に，メモリの[$s2+100]番地のワードデータを読込み
	store word	sw $s1, 100($s2)	メモリの[$s2+100]番地に，$s1のワードデータを書込み
条件分岐	branch on equal	beq $s1, $s2, L	もし、$s1==$s2ならLへ分岐
	branch on not equal	bne $s1, $s2, L	もし、$s1!=$s2ならLへ分岐
	set on less than	slt $s1, $s2, $s3	もし、$s2<$s3なら$s1=1，以外なら$s1=0
無条件ジャンプ	jump	j L	Lにジャンプ
	jump register	jr $s1	$s1の値が示すアドレスにジャンプ

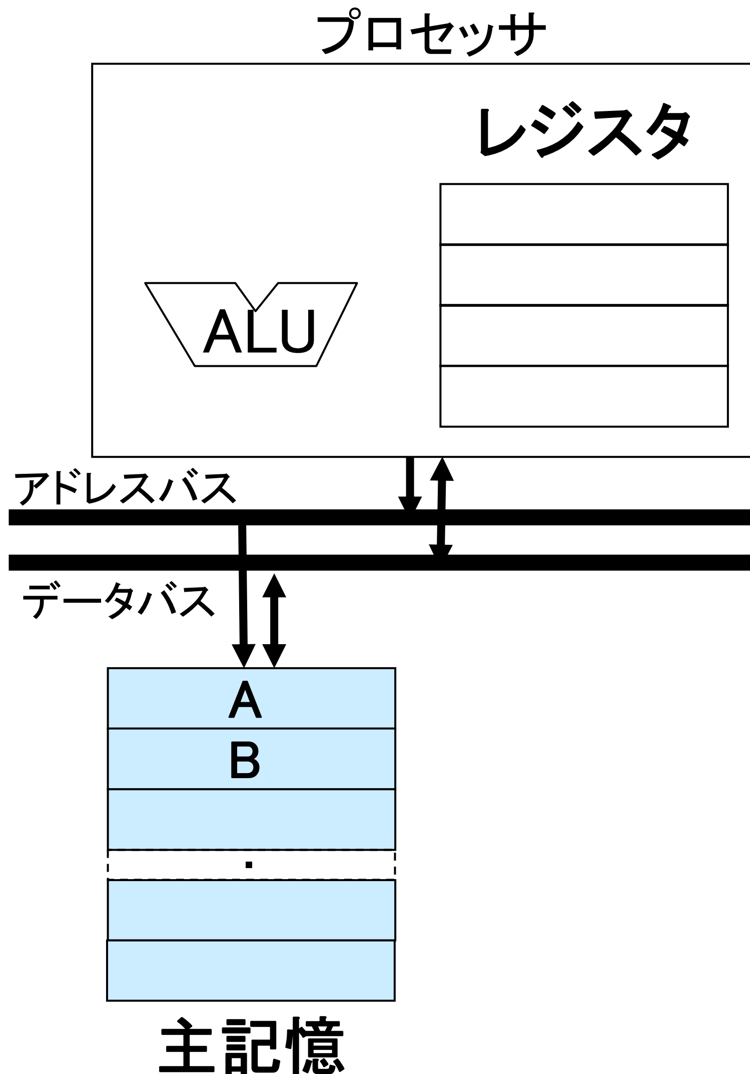
$s1～$s3は汎用レジスタ

プロセッサでの命令実行(1)



- MIPSはロード/ストア・アーキテクチャを採用
  - 主記憶へのアクセスは, ロード命令(読出し)とストア命令(書込み)だけが可能である(ロード/ストア命令に関しては後で説明)
  - 全ての演算はレジスターレジスタ間で行う
  - レジスタとは, プロセッサ内部に搭載された「小容量で高速アクセス可能なメモリ」である

プロセッサでの命令実行(2)



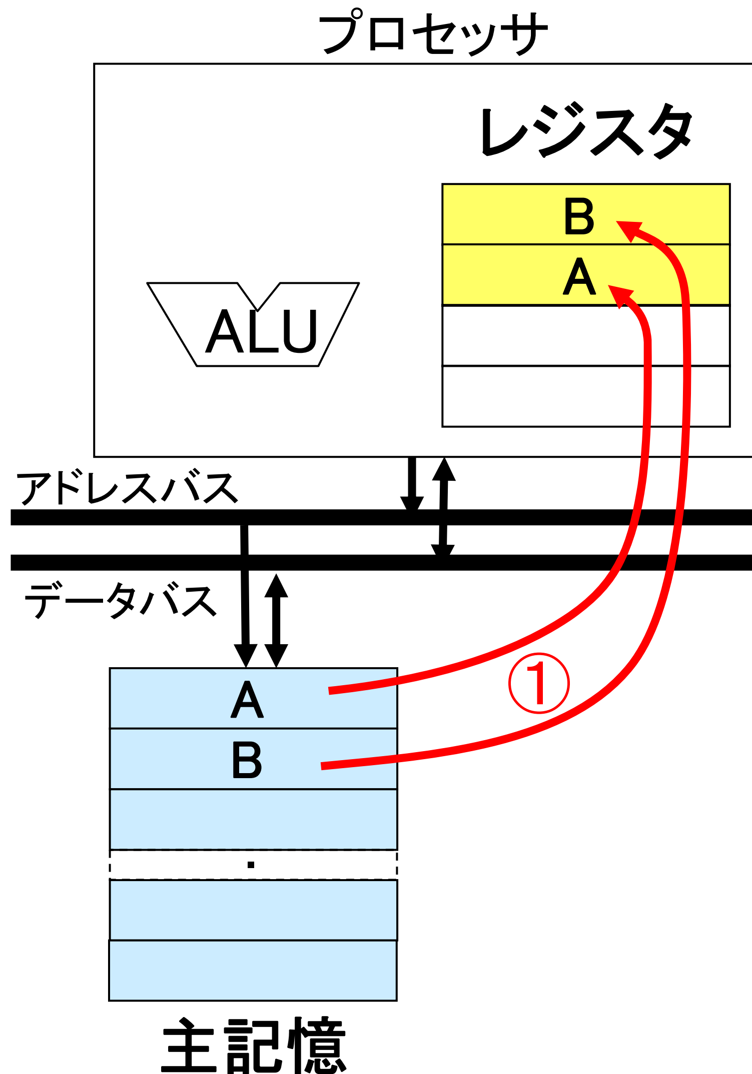
メモリ上の2つのデータに対して加算を行い，結果をメモリに書き戻したい場合

１:メモリからレジスタへデータ読込み
（データ転送:ロード命令）

２:レジスタ間での演算
（算術演算:加算命令）

３:レジスタの値をメモリへ書込み
（データ転送:ストア命令）

プロセッサでの命令実行(2)



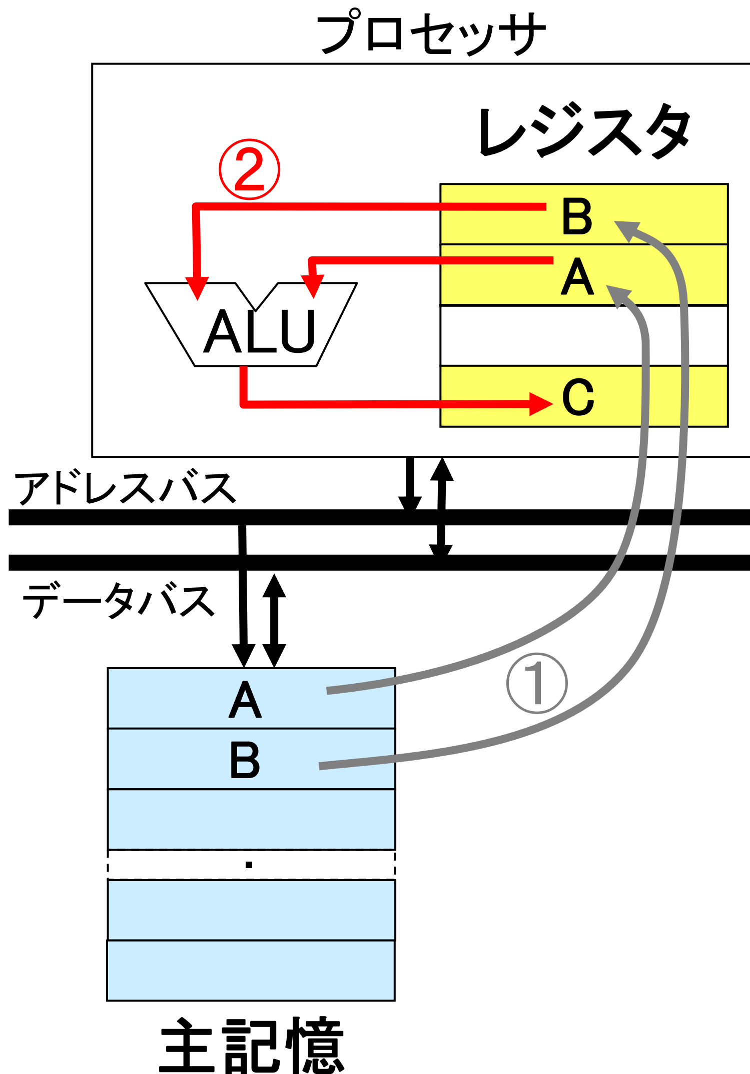
メモリ上の2つのデータに対して演算を行い，結果をメモリに書き戻したい場合

1：メモリからレジスタへデータ読込み
（データ転送：ロード命令）

2：レジスタ間での演算
（算術演算：加算命令）

3：レジスタの値をメモリへ書込み
（データ転送：ストア命令）

プロセッサでの命令実行(2)



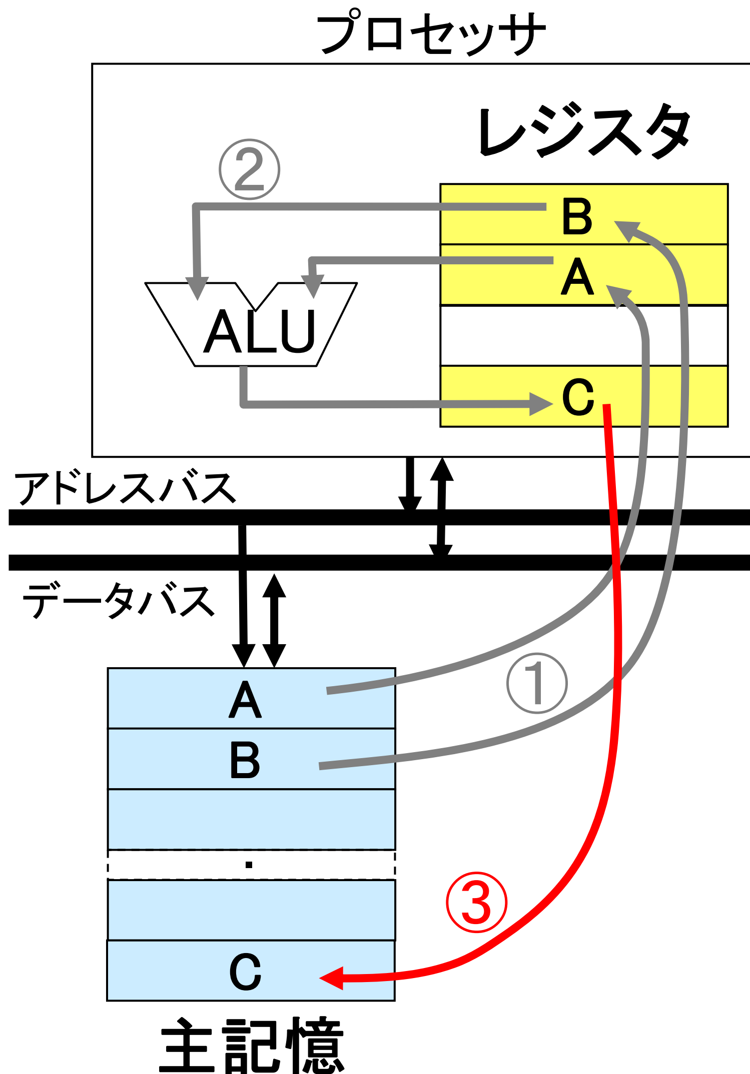
メモリ上の2つのデータに対して演算を行い，結果をメモリに書き戻したい場合

1：メモリからレジスタへデータ読込み
（データ転送：ロード命令）

2：レジスタ間での演算
（算術演算：加算命令）

3：レジスタの値をメモリへ書込み
（データ転送：ストア命令）

プロセッサでの命令実行(2)



メモリ上の2つのデータに対して演算を行い，結果をメモリに書き戻したい場合

1：メモリからレジスタへデータ読込み
（データ転送：ロード命令）

2：レジスタ間での演算
（算術演算：加算命令）

3：レジスタの値をメモリへ書込み
（データ転送：ストア命令）

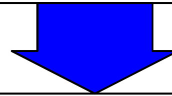
レジスタ（1）

- 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.
- レジスタはプログラミング言語の変数のような使い方をする.
  - レジスタは，通常は主記憶に格納されている頻繁に参照されるデータを一時的に記憶するために使用される.
  - レジスタは，複雑な計算の中間結果など，一時的に使用されるデータを記憶するために使用される.
- 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 = 32bit）
  - 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.

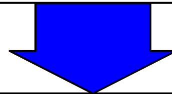
レジスタ(2)

直感的にはレジスタの数を増やせば速くなるが･･･

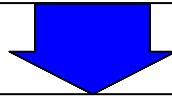
☺レジスタを増やした.



☹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.

設計原則: 小型化は高速化につながる.

MIPSのレジスタオペランド

- 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．
  - プログラム中の変数として使用するレジスタ: $s0, $s1, …, $s7（レジスタ16番から23番に相当）
  - 計算の途中結果など一時変数として使用するレジスタ: $t0, $t1, …, $t7（レジスタ8番から15番に相当）
- レジスタオペランドは，レジスタ名「$△△」をそのまま記述する．

32個のレジスタとその用途

レジスタ番号	略号	用途
0	zero	定数の0
1	at	アセンブラ予約
2	v0	式の評価と関数の結果
3	v1	式の評価と関数の結果
4	a0	引数
5	a1	引数
6	a2	引数
7	a3	引数
8	t0	一時変数
9	t1	一時変数
10	t2	一時変数
11	t3	一時変数
12	t4	一時変数
13	t5	一時変数
14	t6	一時変数
15	t7	一時変数
16	s0	一時変数
17	s1	一時変数
18	s2	一時変数
19	s3	一時変数
20	s4	一時変数
21	s5	一時変数
22	s6	一時変数
23	s7	一時変数
24	t8	一時変数
25	t9	一時変数
26	k0	OS用予約
27	k1	OS用予約
28	gp	ポインタ用
29	sp	ポインタ用
30	fp	ポインタ用
31	ra	戻りアドレス

九州大学工学部電気情報工学科（2006年度）

MIPSの命令（一部）

命令区分	命令	例	意味
算術演算	add	add $s1, $s2, $s3	$s1 = $s2 + $s3
	Subtract	sub $s1, $s2, $s3	$s1 = $s2 − $s3
データ転送	load word	lw $s1, 100($s2)	$s1に，メモリの[$s2+100]番地のワードデータを読込み
	store word	sw $s1, 100($s2)	メモリの[$s2+100]番地に，$s1のワードデータを書込み
条件分岐	branch on equal	beq $s1, $s2, L	もし、$s1==$s2ならLへ分岐
	branch on not equal	bne $s1, $s2, L	もし、$s1!=$s2ならLへ分岐
	set on less than	slt $s1, $s2, $s3	もし、$s2<$s3なら$s1=1，以外なら$s1=0
無条件ジャンプ	jump	j L	Lにジャンプ
	jump register	jr $s1	$s1の値が示すアドレスにジャンプ

$s1～$s3は汎用レジスタ

算術演算：加算命令/減算命令（1）

例）

add $s1, $s2, $s3

レジスタ$s2の値と、レジスタ$s3の値を加算して、レジスタ$s1に格納する

sub $s1, $s2, $s3

レジスタ$s2の値からレジスタ$s3の値を減算して、レジスタ$s1に格納する

プロセッサ

レジスタ

ALU

+/−

s1

s2

s3

s4

アドレスバス

データバス

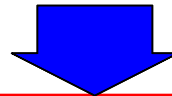
算術演算:加算命令/減算命令(2)

アセンブリ言語の例

add	$s1, $s2, $s3	# $s2 + $s3 の結果を $s1 に格納
sub	$s4, $s1, $s2	# $s1 – $s2 の結果を $s4 に格納

ニーモニック　　オペランド　　注釈(コメント)

MIPS の算術／論理演算命令はどれも上記の3オペランド形式



設計原則: 単純性は規則性につながる.

アセンブリ言語プログラムを書いてみよう！

問題：以下に示すように，5つの変数a, b, c, d, eを含むC言語プログラム（の一部）がある．これをMIPSアセンブリ言語に変換せよ．ただし，各変数のレジスタ割当ては下表に従う．

```
a = b + c;
d = a - e;
```

変数名	レジスタ（略号）
a	s1
b	s2
c	s3
d	s4
e	s5

解答：

```
add  $s1,  $s2,  $s3   #a = b + cのコード
sub  $s4,  $s1,  $s5   #d = a - eのコード
```

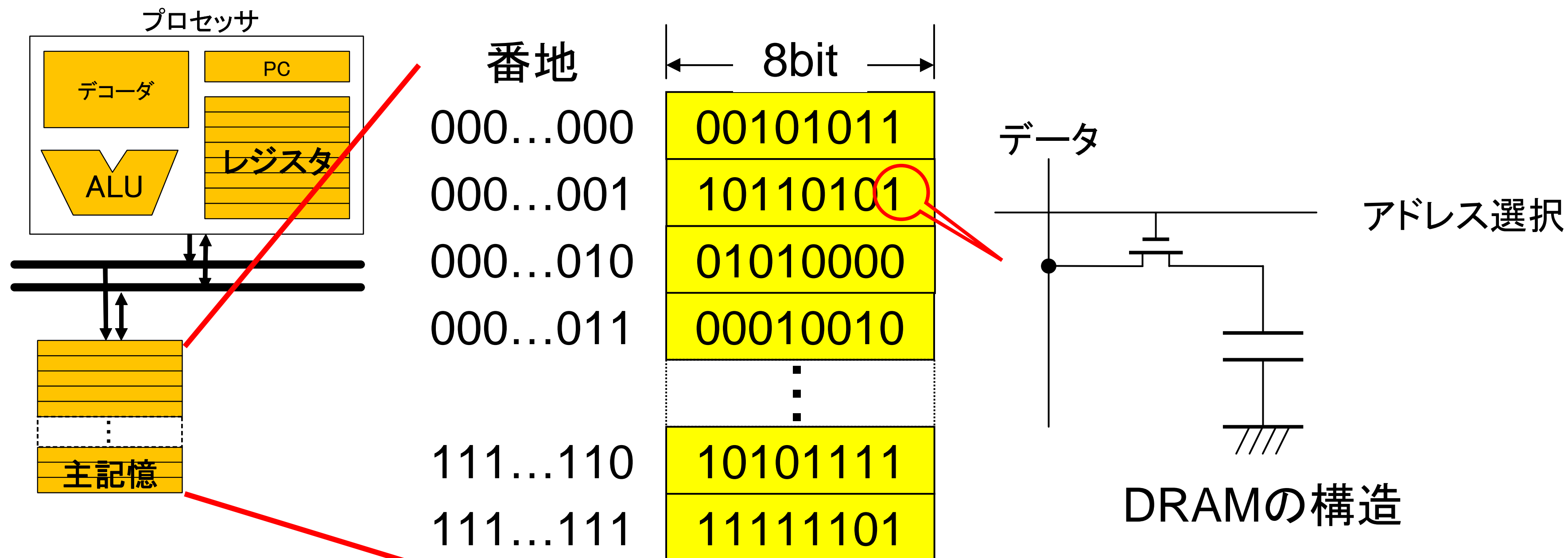
MIPSの命令（一部）

命令区分	命令	例	意味
算術演算	add	add $s1, $s2, $s3	$s1 = $s2 + $s3
	Subtract	sub $s1, $s2, $s3	$s1 = $s2 − $s3
データ転送	load word	lw $s1, 100($s2)	$s1に，メモリの[$s2+100]番地のワードデータを読込み
	store word	sw $s1, 100($s2)	メモリの[$s2+100]番地に，$s1のワードデータを書込み
条件分岐	branch on equal	beq $s1, $s2, L	もし、$s1==$s2ならLへ分岐
	branch on not equal	bne $s1, $s2, L	もし、$s1!=$s2ならLへ分岐
	set on less than	slt $s1, $s2, $s3	もし、$s2<$s3なら$s1=1，以外なら$s1=0
無条件ジャンプ	jump	j L	Lにジャンプ
	jump register	jr $s1	$s1の値が示すアドレスにジャンプ

$s1～$s3は汎用レジスタ

主記憶(1)

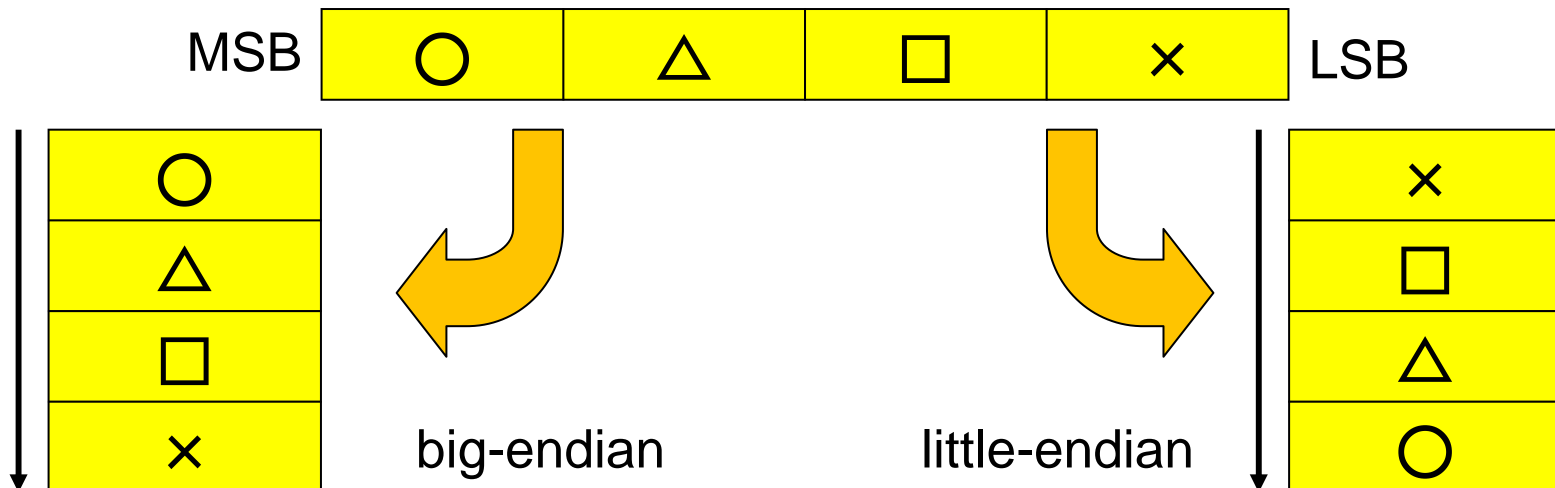
- 主記憶は，順番に**番地**(アドレス)が振られた，一列に並んだセルである.
- 今日では通常，各々のセルは1バイト，すなわち8ビットの二進数を記憶する.
- 主記憶は，番地がインデックスになる，巨大な8ビット幅の一次元配列と考えられる.
- 今日は主記憶には**DRAM**(**Dynamic RAM**)が使用されている.



主記憶(2)

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する.

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



主記憶（3）

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．

整列化制約がないと･･･

データ転送：ロード/ストア命令

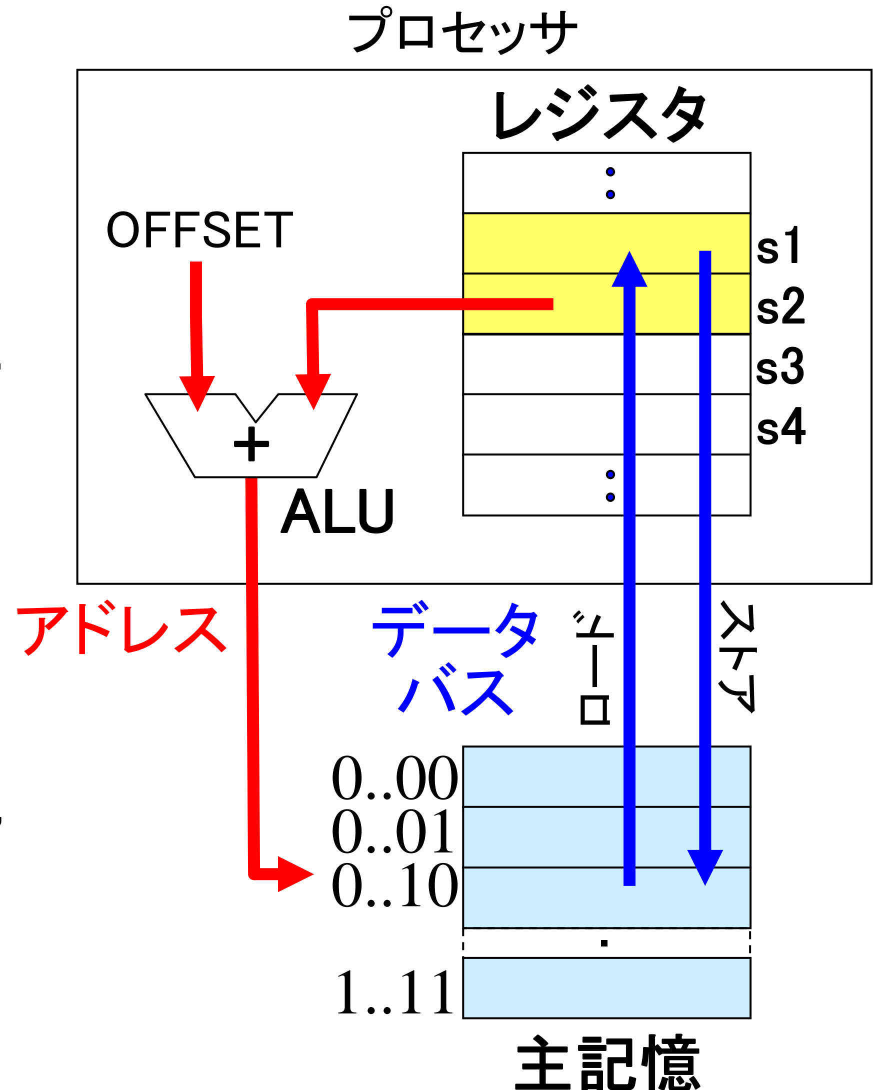
例）

lw $s1, OFFSET($s2)

レジスタ$s2の値にOFFSETを加えてメモリ・アクセス用アドレスを得る．このアドレスで指定されるメモリ内データを読出し，レジスタ$s1に格納する（ロード）．

sw $s1, OFFSET($s2)

レジスタ$s2の値にOFFSETを加え，メモリ・アクセス用アドレスを得る．このアドレスで指定されるメモリ領域に対しレジスタ$s1の値を書き込む（ストア）．



MIPS のメモリオペランド

- 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**ベースレジスタ**と呼ぶ．
  - n には，符号つき整数が指定でき，これを**オフセット**と呼ぶ．
- MIPS では，算術演算命令（add，sub など）でメモリオペランドは指定できない．（＝演算に使うデータはかならずレジスタに置かなければならない!）

メモリオペランドを使用する命令の例:

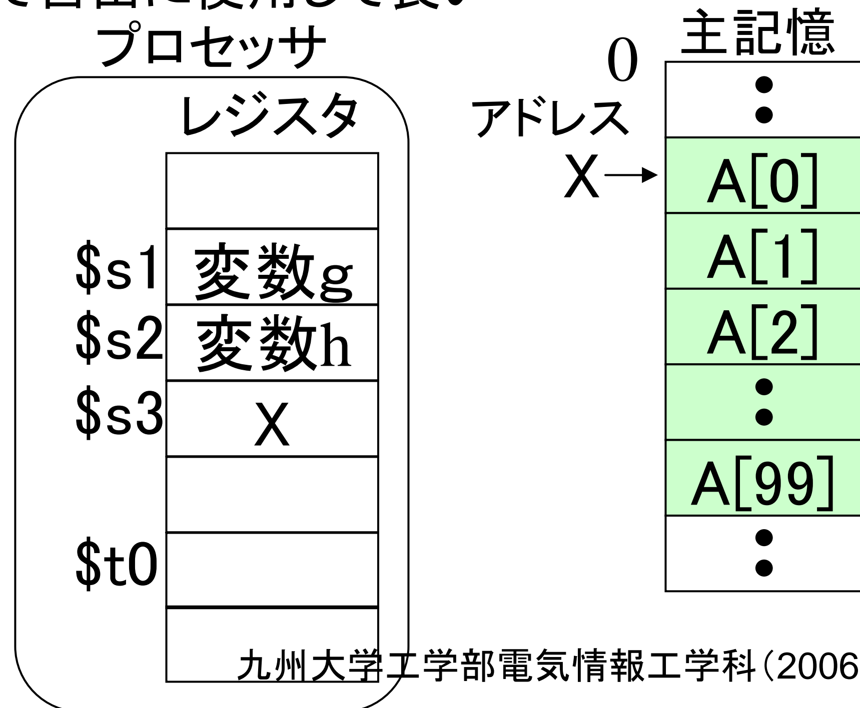
```
lw    $t0, 32($s0)    # ($s0 + 32) 番地の内容を $t0 に格納
sw    $t0, –8($s0)    # ($s0 – 8) 番地に $t0 を格納
```

アセンブリ言語プログラムを書いてみよう！

問題：以下に示すC言語プログラム（の一部）がある．★で示した代入文をMIPSアセンブリ言語に変換せよ．ただし，

- 配列Aは100語（1語は32ビット）からなり，その開始アドレス（ベースアドレス）はレジスタ$s3に格納されているとする
- 変数gはレジスタ$s1に，変数hはレジスタ$s2に格納されているとする
- バイトアドレス方式を前提とする
- レジスタ$t0は一時変数用として自由に使用して良い

```
int A[100];
int g, h;
      :
g = h + A[2];   /* ★ */
```



アセンブリ言語プログラムを書いてみよう！

問題：以下に示すC言語プログラム（の一部）がある．★で示した代入文をMIPSアセンブリ言語に変換せよ．

解答例

Step1:配列データA[2]をレジスタへロード

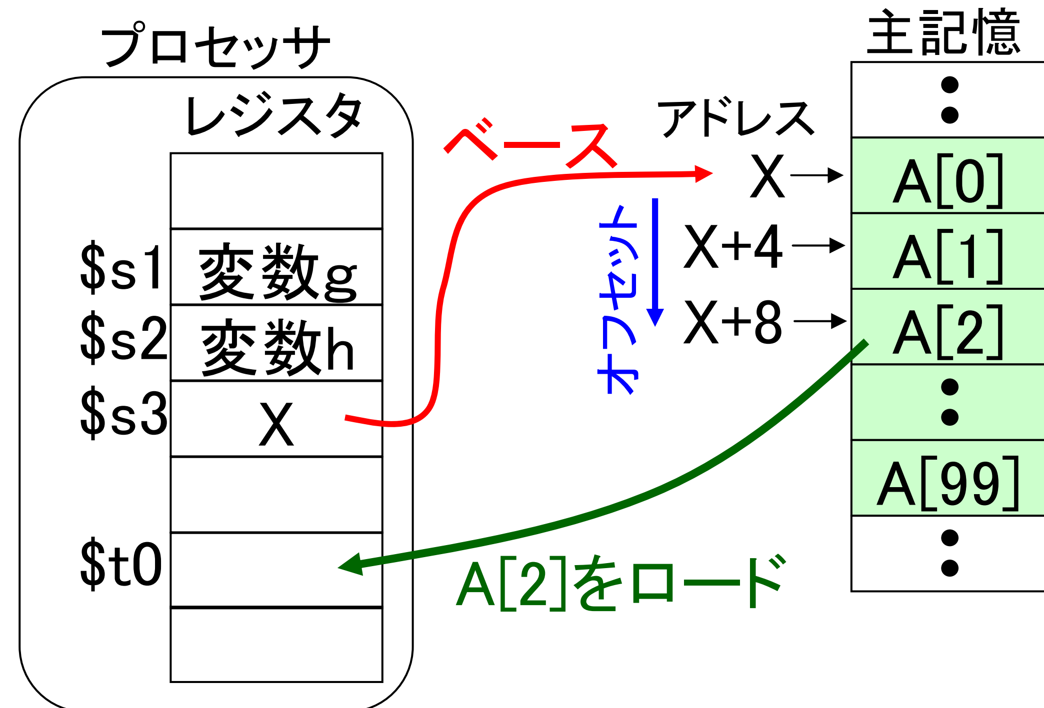
lw $t0, 8 ($s3)

バイトアドレス方式に注意！
（1語は4バイト）

Step2:hとA[2]の加算

add $s1, $s2, $t0

g = h + A[2]; /* ★ */




```
lw  $t0, 8 ($s3)
add $s1, $s2, $t0
```

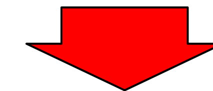
命令表現(1)

高水準プログラミング言語
(high-level programming language)

```
if (y == 1)
    c = a + b;
else
    c = a - b;
```

命令セット

コンパイラ
(compiler)



どのようにして「各命令」を
2進数で表現するのか？

機械語
(CPUへの命令)

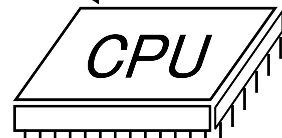
```
1001010010101100
0000010110111100
1110011110100111
```

2進表現

「命令のシーケンス」と
してプログラムを表現

```
Target:
    lw $4, 0($1)
    lw $5, 4($1)
    add $2, $4, $5
```

アセンブリ言語



命令表現(2)

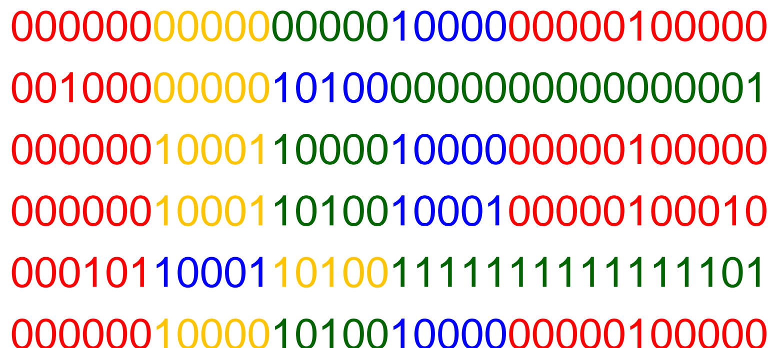
今日のコンピュータは全ての情報を2進数で表現する.

命令も例外ではない!

```
# 1 から $s1 までの和を $s0 に納めるプログラム(ただし、$s1の値は2以上とする)
        add     $s0, $zero, $zero
        addi    $s4, $zero, 1
L1:     add     $s0, $s1, $s0
        sub     $s1, $s1, $s4
        bne     $s1, $s4, L1
        add     $s0, $s0, $s4
```

下記の情報を2進数に符号化:

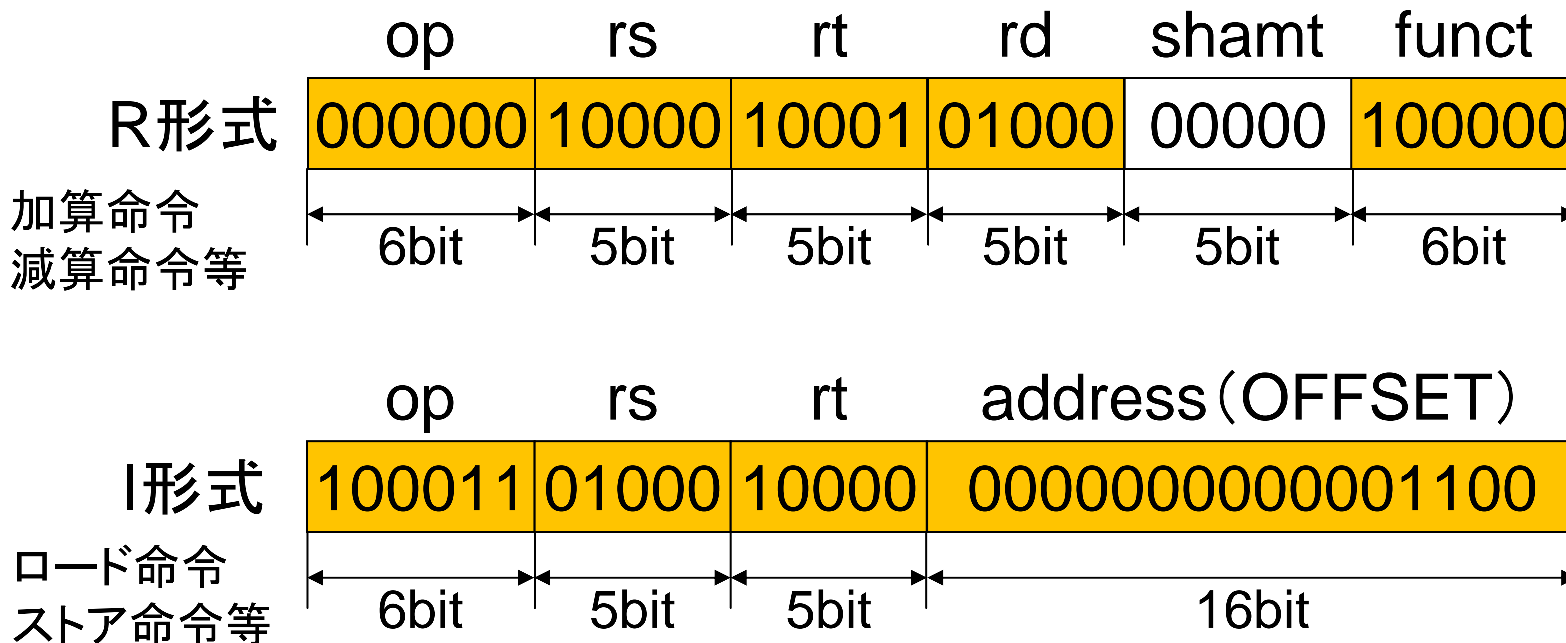
- 命令の種類
- 読み書きされるレジスタ
- 読み書きされる主記憶の番地
- 一緒に計算される値 *etc.*



命令表現(3)

命令形式: 命令語のフィールド構成.

- MIPS は命令語を32bit 幅で統一している.
- MIPS の命令形式は, R形式, I形式, J形式(後述)の3種類がある.
- どの命令形式かは, op フィールド(**命令操作コード**)で判別できる.



命令表現(4)

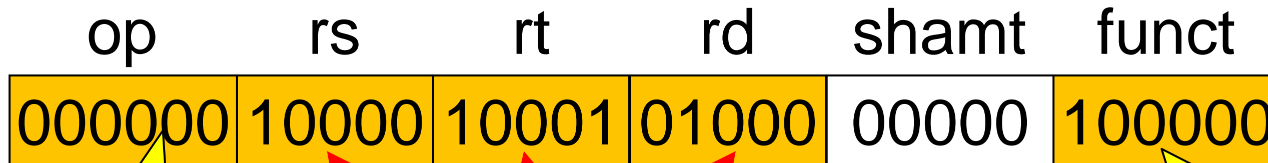
例）　add　$t0, $s0, $s1

命令語に符号化すべき情報:

- 命令の種類: add
- ディスティネーションオペランド: $t0
- ソースオペランド: $s0, $s1

命令表現(5)

例）　add　$t0, $s0, $s1



$s0　$s1　$t0

加算または減算を意味する.

100000のときは加算,
100010のときは減算を意味する.

01000	$t0	10000	$s0
01001	$t1	10001	$s1
…	…	…	…
01111	$t7	10111	$s7

命令表現(6)

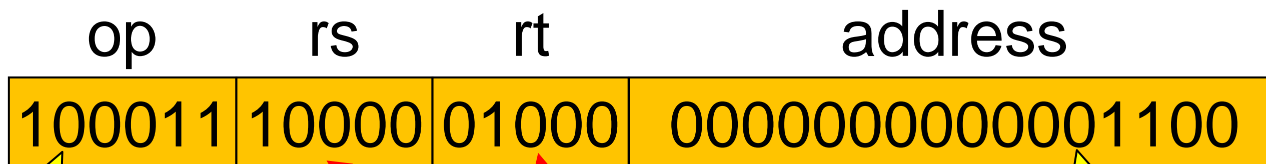
例)　　lw　　$t0, 12($s0)

命令語に符号化すべき情報:

- 命令の種類: lw
- ディスティネーションオペランド: $t0
- ベースレジスタ: $s0
- オフセット: +12

命令表現(7)

例）　lw　$t0, 12($s0)



lw　$s0　$t0　+12

100011のときは lw,
101011のときは sw
を意味する.

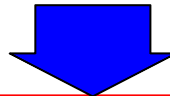
16bit符号つき整数
（2の補数表現）
–32768～+32767

01000	$t0	10000	$s0
01001	$t1	10001	$s1
…	…	…	…
01111	$t7	10111	$s7

命令表現(8)

命令形式設計のポイント:

- 命令形式の種類の数は少なく!
- 命令形式には規則性を!



ハードウェアの単純化と高速化

ベースレジスタの番地より –32768 以前, 32767 以降のアドレスにアクセスしたいときは?

無理

設計原則: すぐれた設計には適度な妥協が必要である.