

情報通信概論: プロセッサ入門

担当：一色剛

工学院情報通信系

東京工業大学

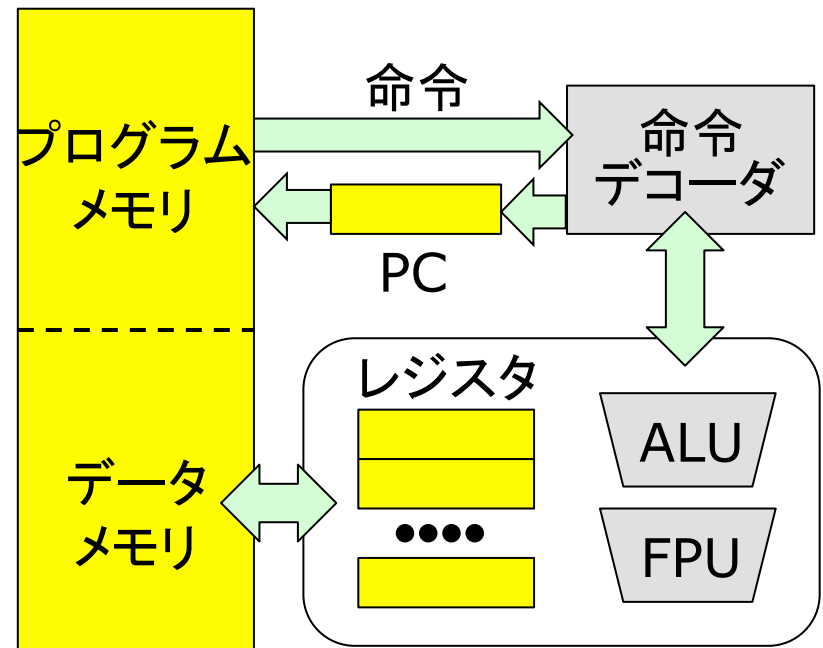
issiki@ict.e.titech.ac.jp

講義概要

1. プロセッサ構成要素：メモリ、レジスタ、プログラムカウンタ, ALU (算術論理演算器), FPU (浮動小数点演算器)
2. 命令実行制御
3. 論理回路による算術演算
4. 命令セット：CISC と RISC
 - 命令機能：データ転送(load, store), 演算 (add/sub/mult/etc), プログラム制御 (branch, call/return)
 - 命令フォーマット：機械語、アセンブリ言語, レジスタ転送記述
5. 命令セットの例：x86, MIPS

1. プロセッサ構成要素

- メモリ：プログラム、データ
- レジスタ：データ一時格納
- プログラムカウンタ (PC)：実行命令の格納場所
- 命令デコーダ：命令機能の特定
 - データ転送、演算
 - プログラム制御
- ALU (算術論理演算器)：
 - 四則演算(+, -, *, /)
 - 論理演算(&, |, ^)
 - シフト演算(<<, >>)
- FPU (浮動小数点演算器)：
 - 四則演算
 - 整数変換

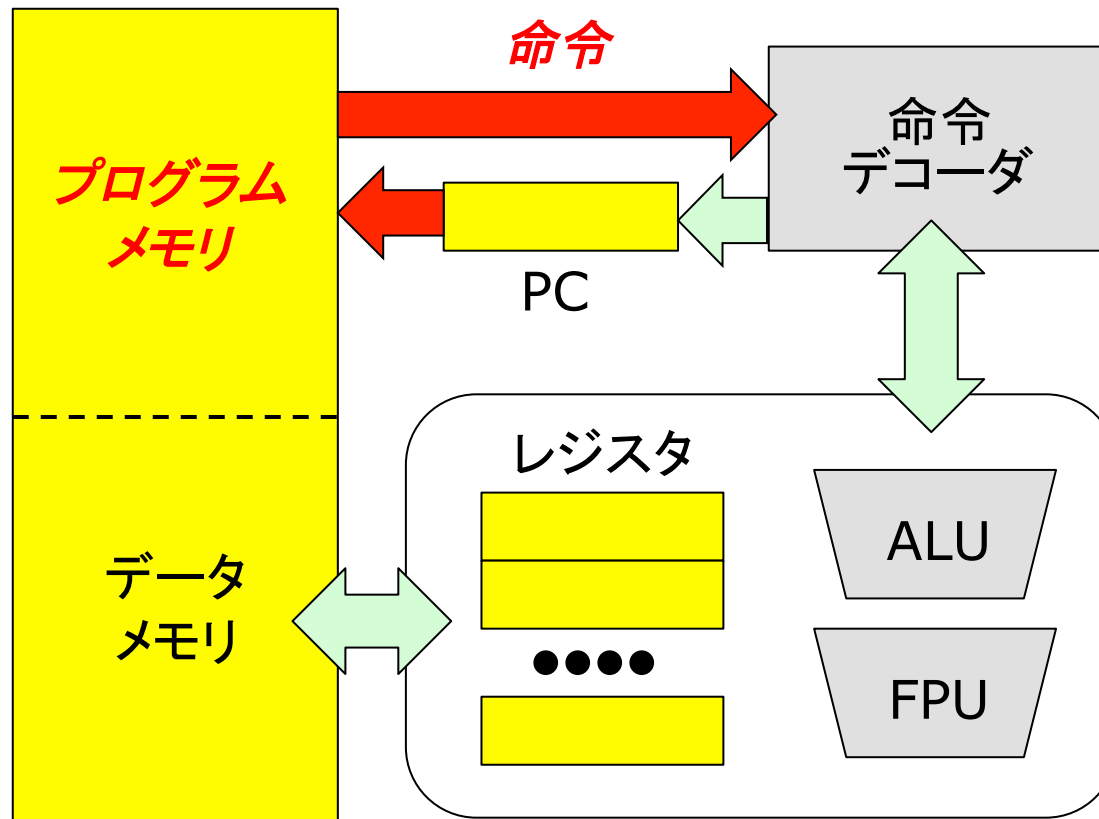


2. 命令実行制御

- 1) 命令フェッチ : PCで指定されたプログラムメモリの格納場所から命令を読み出す
- 2) 命令デコード : 命令機能を解読する
- 3) 命令実行 :
 - a. データ転送 : メモリからデータをレジスタにロード
 - b. 演算 : ALU/FPU 命令の実行
 - c. プログラム制御 : 次のPCを計算 (ジャンプ、条件分岐、関数呼出し、関数リターン)
- 4) 結果保存 : レジスタ・メモリに格納
- 5) PC更新

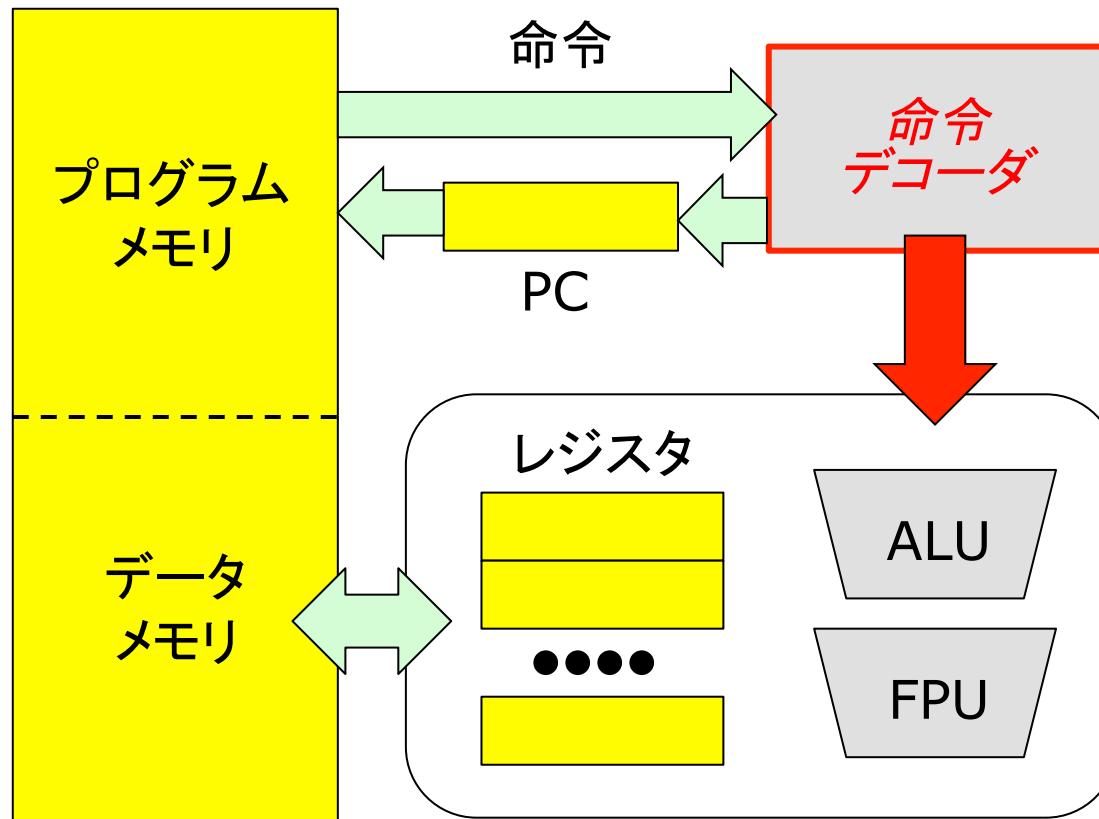
2. 命令実行制御

- 1) 命令フェッチ：PCで指定されたプログラムメモリの格納場所から命令を読み出す



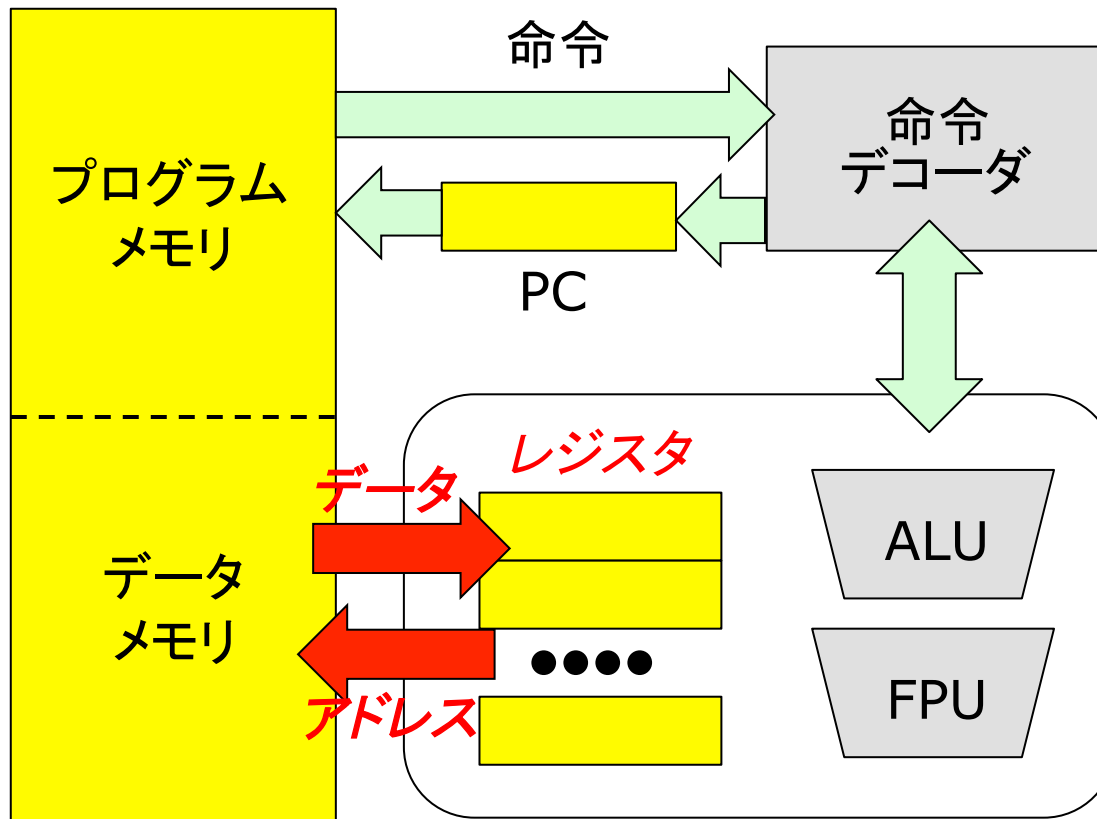
2. 命令実行制御

2) 命令デコード：命令機能を解読する



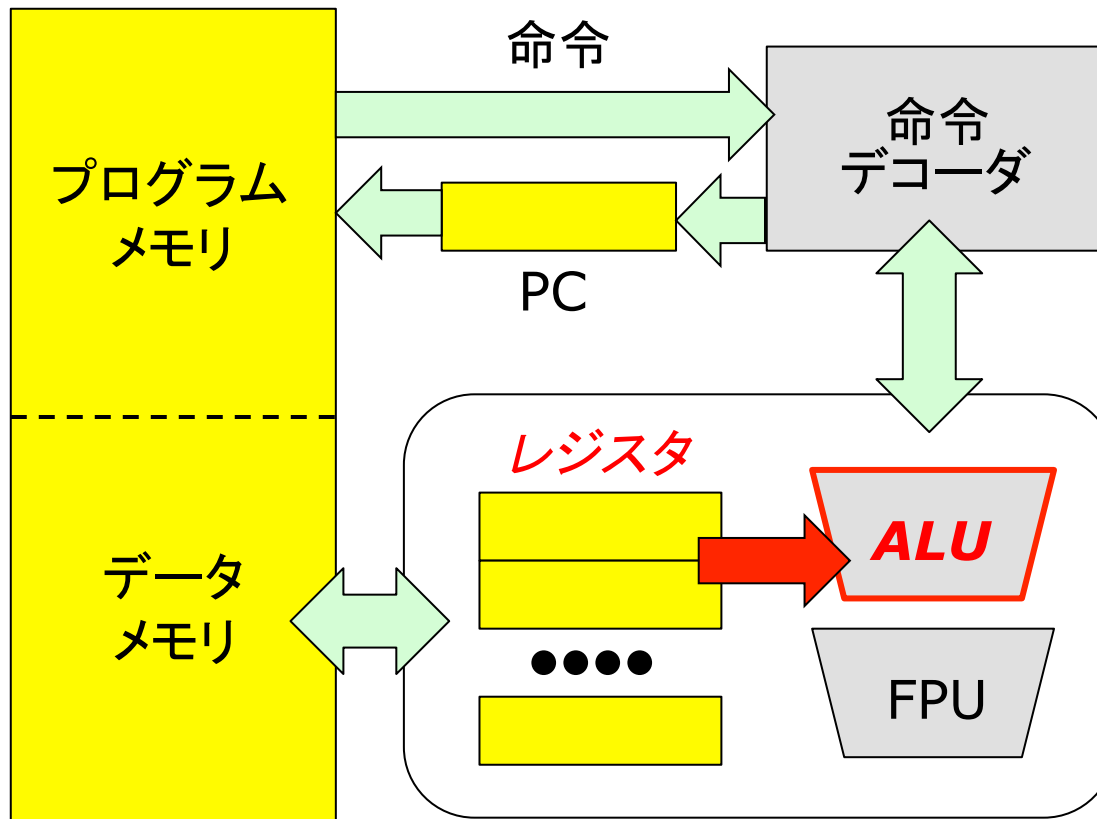
2. 命令実行制御

3) 命令実行 : (a) メモリからデータをレジスタにロード



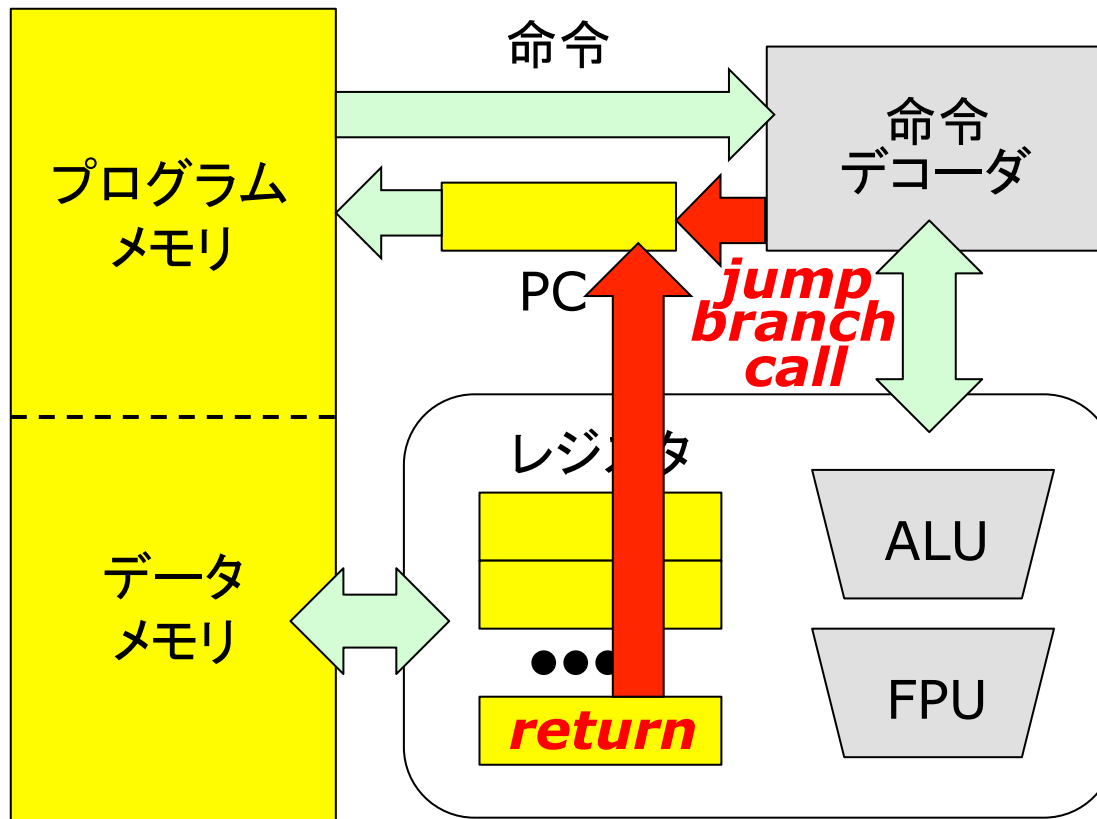
2. 命令実行制御

3) 命令実行 : (b) 演算 : ALU/FPU 命令の実行



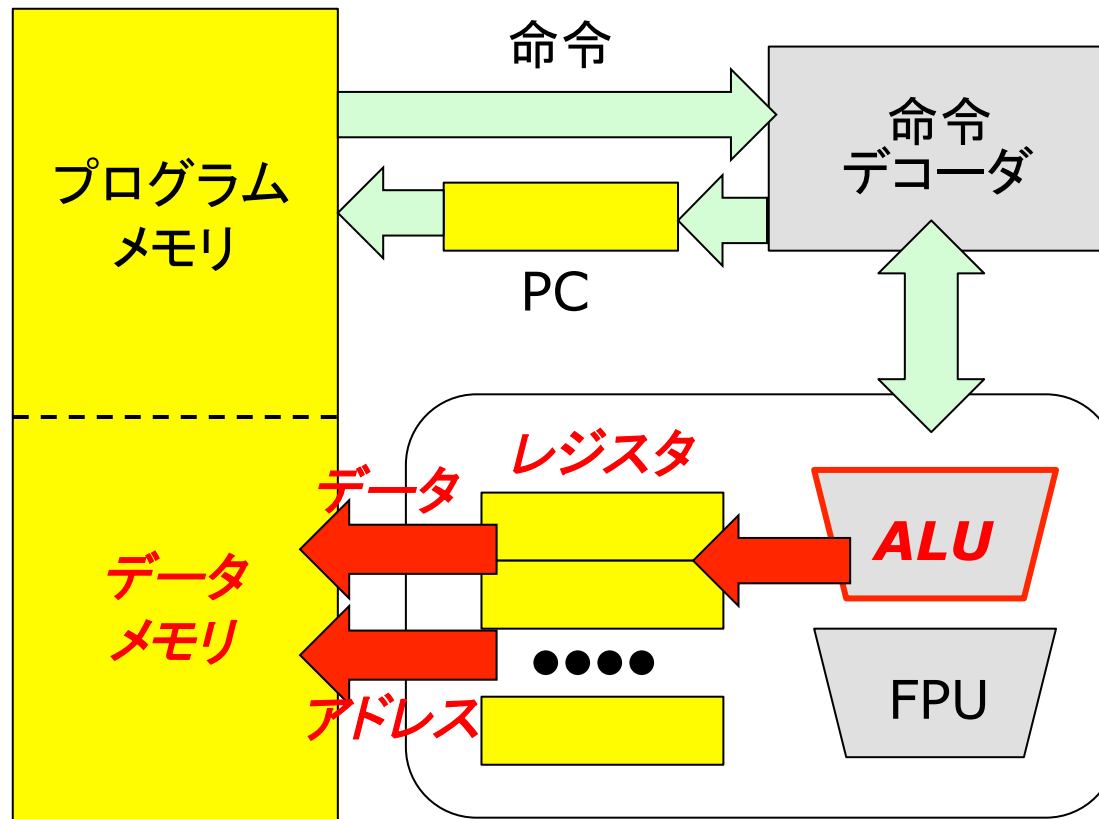
2. 命令実行制御

3) 命令フェッチ：(c) 次のPCを計算 (ジャンプ、条件分岐、関数呼出し、関数リターン)



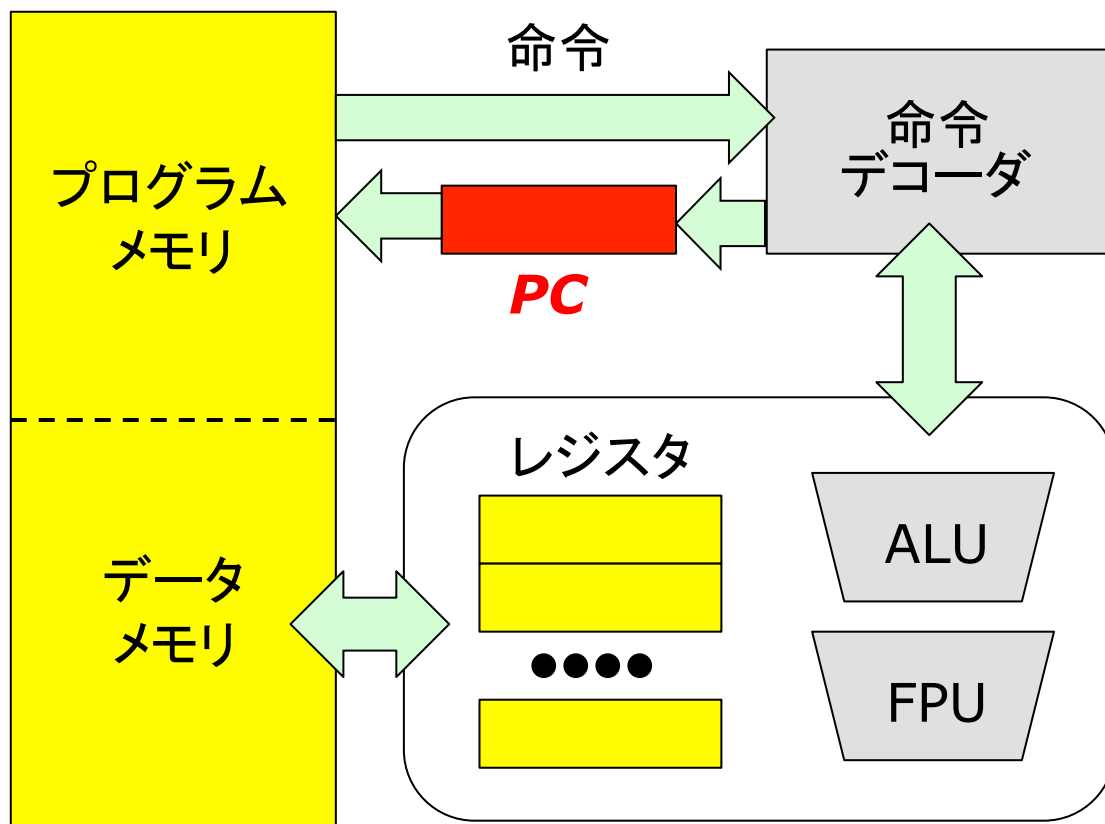
2. 命令実行制御

4) 結果保存：レジスタ・メモリに格納



2. 命令実行制御

5) PC更新



3. 論理回路による算術演算

2進データ表現

- 符号なし整数 (unsigned)

$$value = \sum_{i=0}^{N-1} b_i \cdot 2^i \quad (b_i : i^{\text{th}} \text{ bit})$$

- 符号付き整数 (signed : 2の補数)

$$value = -b_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} b_i \cdot 2^i$$

- 最上位ビット (most significant bit: MSB) : “sign” bit

sign = 1 : 負整数

sign = 0 : 非負整数

2進	unsigned	signed
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	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

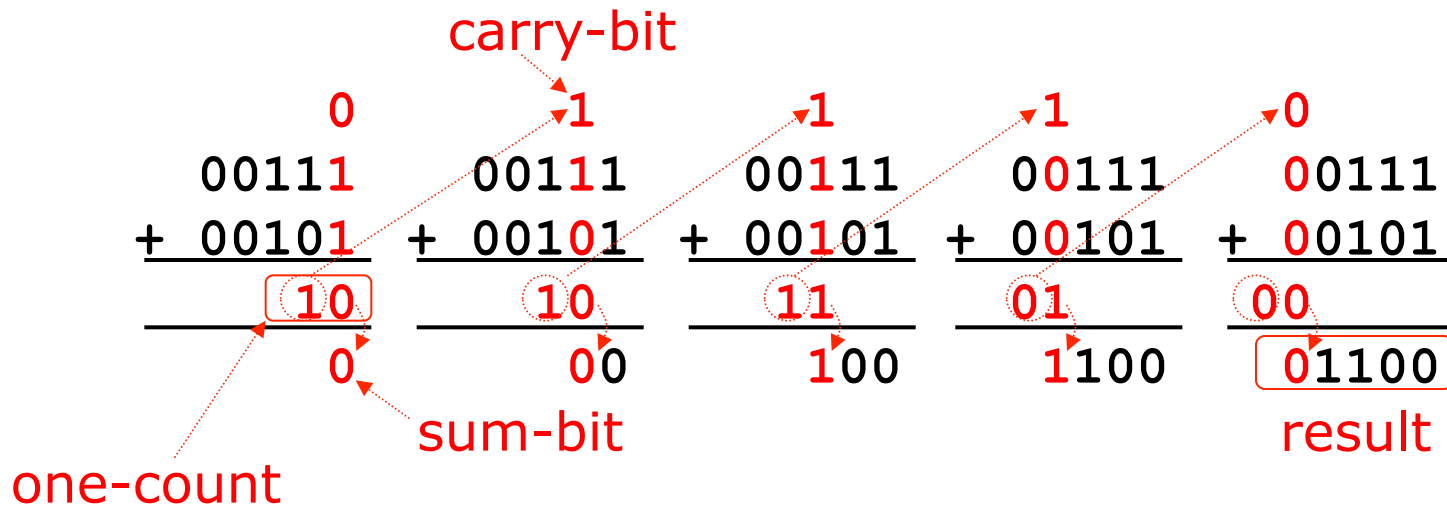
2進数の計算法 (Computer Arithmetic)

- 2進数の計算は10進数の「筆算」と同じ原理
 - 2進数の各桁は0か1なので、計算機に好都合
- 2進数計算は論理回路の主要な機能
(computer arithmetic)

$\begin{array}{r} 7 \quad 00111 \\ + 5 \quad 00101 \\ \hline 12 \quad 01100 \end{array}$	$\begin{array}{r} 12 \quad 01100 \\ \times 13 \quad 01101 \\ \hline 36 \quad 01100 \\ 12 \quad 00000 \\ \hline 156 \quad 01100 \end{array}$	$\begin{array}{r} 22 \\ 5 \overline{) 114} \\ \underline{10} \\ 14 \\ \underline{10} \\ 4 \end{array}$
$\begin{array}{r} 7 \quad 00111 \\ - 5 \quad 00101 \\ \hline 2 \quad 00010 \end{array}$	$\begin{array}{r} 01100 \\ \times 01101 \\ \hline 01100 \\ 00000 \\ \hline 010011100 \end{array}$	$\begin{array}{r} 00010110 \\ 101 \overline{) 01110010} \\ \underline{101} \\ 01000 \\ \underline{101} \\ 0111 \\ \underline{101} \\ 100 \end{array}$

2進数加算

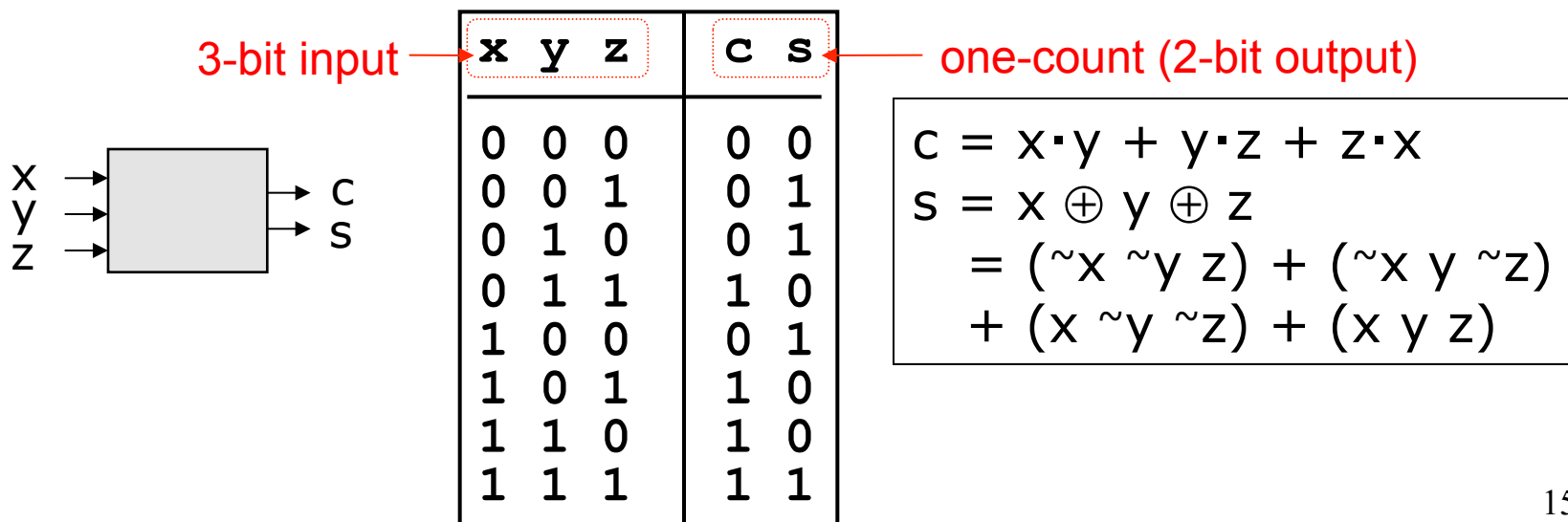
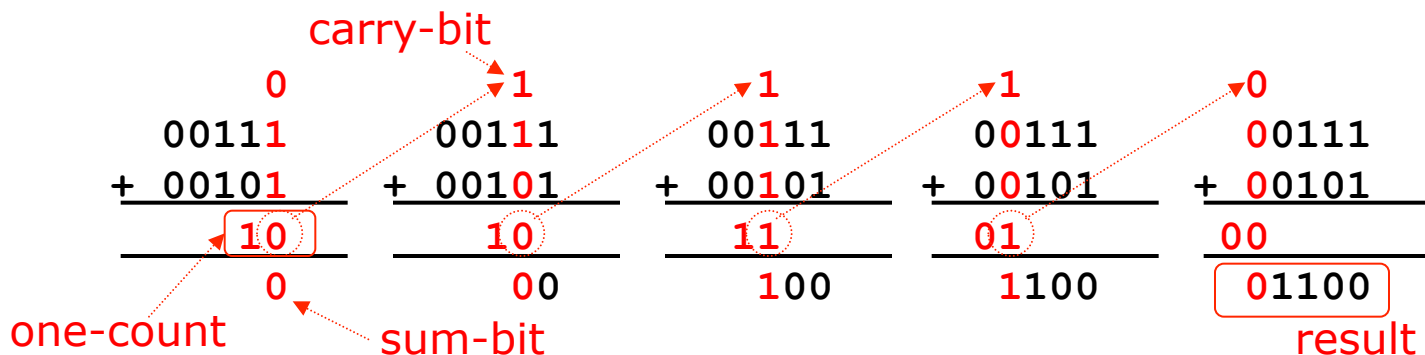
- 各ビットで (最下位ビット(左)から最上位ビット(右)へ):
 - 1の個数(“one-count”)を2進数で表現
 - “out-count”の上位ビット: 次のビットのキャリー(桁上げ)



$$\begin{array}{rcl}
 00111 & + & 00101 \\
 7 & + & 5 \\
 \hline
 & = & 12
 \end{array}$$

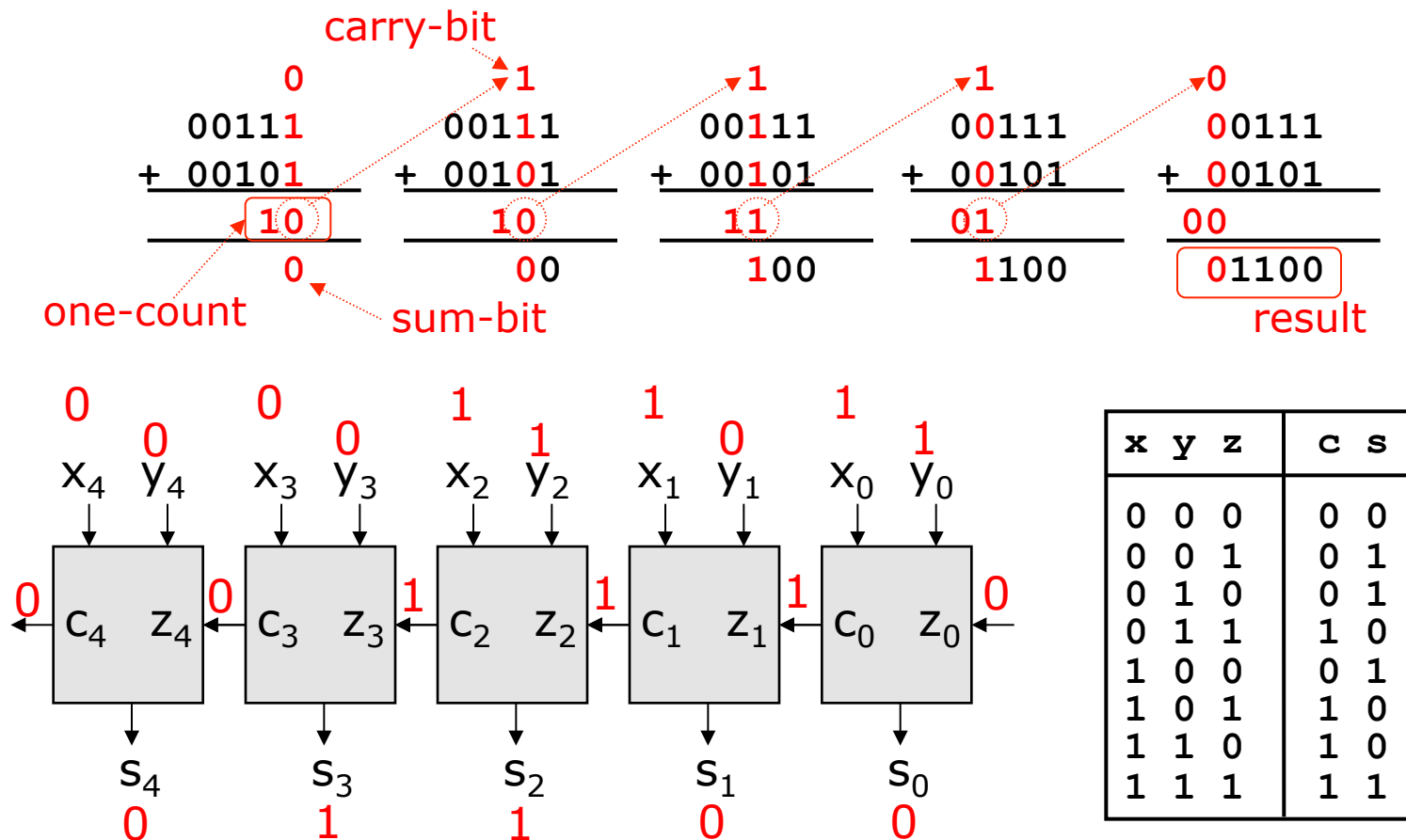
1ビット加算器 (Full Adder)

- 3つのビット入力の1の個数を計算する論理回路



Nビット加算器

- Nビット加算器：N個の1ビット加算器を縦続接続する



2進数減算

- $Z = X - Y$

- Yの各ビットを反転(論理否定)する ($Y \rightarrow Y'$)

- $Y + Y' = 2^N - 1$

- $Y = Y' + 1 - 2^N$

- $Z = X + Y' + 1 - 2^N$

Ex: $00101 + 11010 = 11111$

- 2^N の項はNビット表現では無視できる

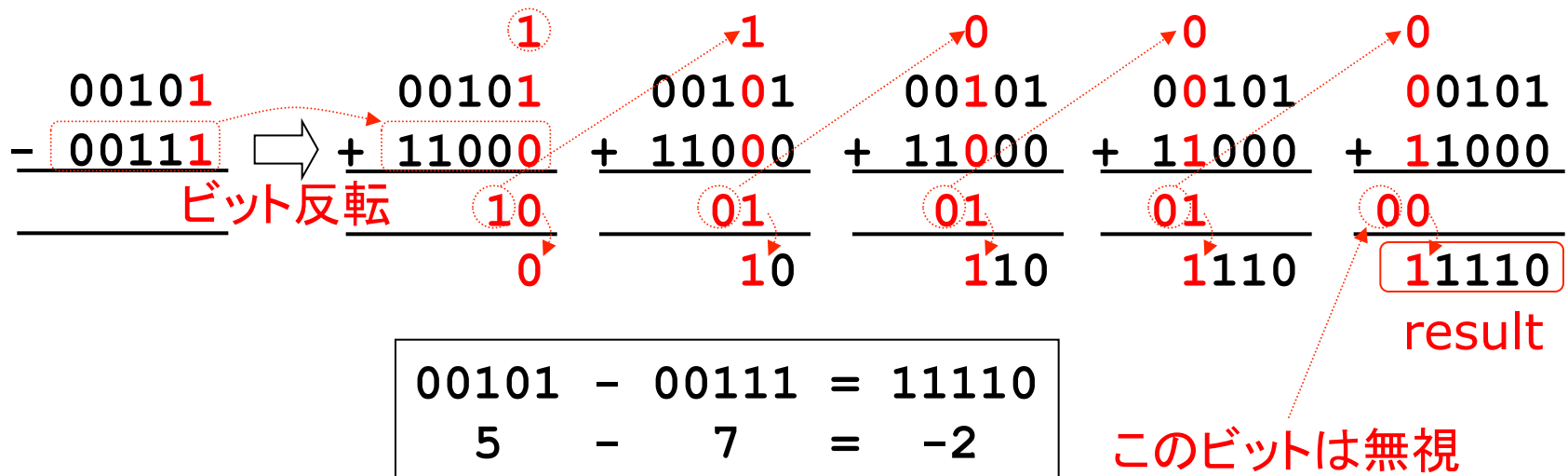
		1	1	1	1	1
00111	00111	00111	00111	00111	00111	00111
- 00101	+ 11010	+ 11010	+ 11010	+ 11010	+ 11010	+ 11010
ビット反転		10	11	10	10	10
	0	10	010	0010	0010	00010
						result

00111	-	00101	=	00010
7	-	5	=	2

このビットは無視

2進数減算

- 減算結果が負の場合は、2進数表現となる



浮動小数点表現 (IEEE 754 規格)

- 単精度 (32-bit, 4-byte)
 - 符号(**S**ign:1-bit), 指数部(**E**xponent:8-bit), 仮数部(**F**raction:23-bit → 符号なし整数)
 - 指数部**E**は -127 ($2^8 - 1$)のオフセット値
 - 正規化数 (通常の場合)
 - 絶対値の範囲: $[2^{-126}, 2^{127}]$ (“0.0”は含まれない)
 - 仮数部**F**の最上位ビットは常に ‘1’ (省略される)
 - $\text{normalized_num} = (-1)^{\mathbf{S}} * 2^{(\mathbf{E} - 127)} * (1 + \mathbf{F} * 2^{-23})$
1.00 → (0 01111111 000000000000000000000000)
1.50 → (0 01111111 100000000000000000000000)
2.25 → (0 10000000 001000000000000000000000)
- 倍精度 (64-bit, 8-byte)
 - 符号(**S**:1-bit), 指数部(**E**:11-bit), 仮数部(**F**:52-bit)
 - $\text{normalized_num} = (-1)^{\mathbf{S}} * 2^{(\mathbf{E} - 1023)} * (1 + \mathbf{F} * 2^{-52})$

浮動小数点表現 (IEEE 754 規格)

- 正規化数以外

- 非正規化数：絶対値が非常に小さい場合 ($< 2^{-126}$)
 - $E = 0$: 2^{-126} の指数部を示す
 - 仮数部 F の最上位ビットは常に '0'
 - $\text{denormalized_num} = (-1)^S * 2^{-126} * (F * 2^{-23})$
- “Zero” (2種類ある)
 - $E = 0, F = 0$

$S = 0$	$\rightarrow +0$
$S = 1$	$\rightarrow -0$
- “Inf” (infinity)
 - $E = 255, F = 0$

$S = 0$	$\rightarrow +\infty$
$S = 1$	$\rightarrow -\infty$
- “NaN” (not-a-number)
 - $E = 255, F \neq 0$
 - $x / 0.0$ is “NaN” (divided by 0)

4. 命令セット

1) **CISC:** Complex Instruction-Set Computer (Intel x86)

- 命令語の長さが可変 : 1 byte ~ 17 bytes → 命令デコーダが複雑
- 演算オペランド(演算対象の入力データ)はメモリの場合もある
- 演算結果はメモリに格納される場合もある

2) **RISC:** Reduced Instruction-Set Computer (MIPS, ARM)

- 命令語の長さが固定 : 2 bytes, 4 bytes → 命令デコーダが単純
- メモリアクセス命令 : ロード(読出し)、ストア(書込み)だけ
- 演算オペランドと演算結果格納場所: レジスタ

命令機能

1) データ転送 (**load/store**)

- メモリアドレスの指定方法(アドレスモード)
 - 絶対アドレス
 - Base-reg + オフセット値
 - Base-reg + index-reg * スケール値 + オフセット値

2) 演算 (**add/sub/mult/div/...**)

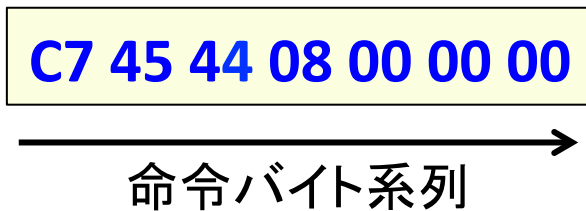
- オペランドの指定方法
 - RISC : レジスタ、即値(定数)
 - CISC : メモリ、レジスタ、即値

3) プログラム制御 (**branch, call/return**)

- 分岐条件の指定方法 :
- ジャンプ先アドレスの指定方法
 - 絶対アドレス
 - PC相対アドレス

命令フォーマット (x86の例)

1) 機械語

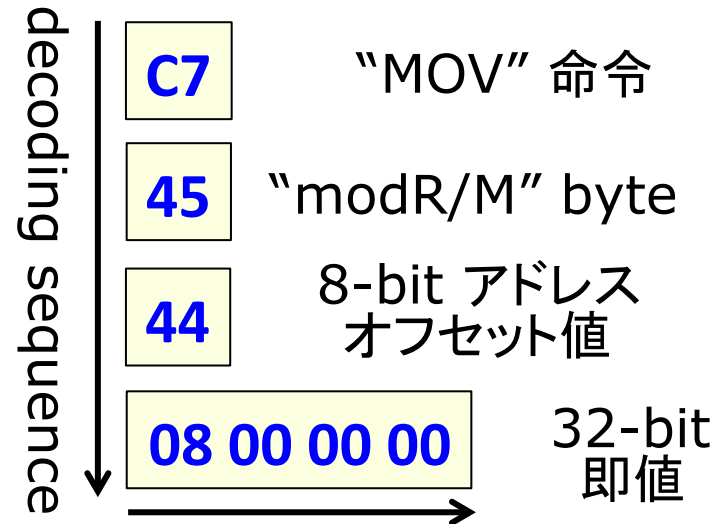


2) アセンブリ言語

mov dword ptr [rbp+44h],8

3) レジスタ転送記述

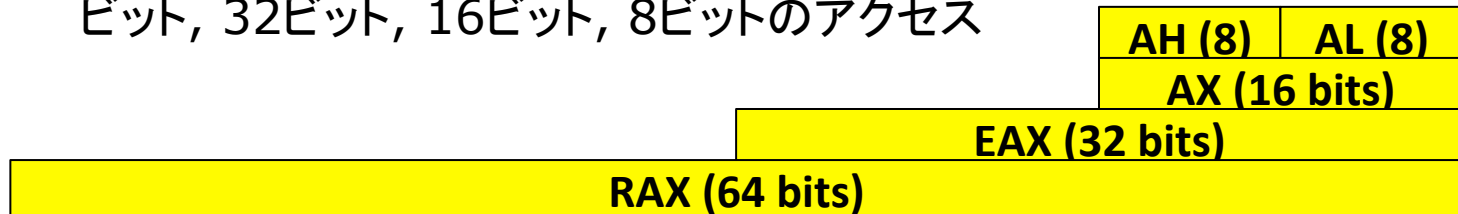
$M_{32}[rbp+44h] \leftarrow 00000008h$



Intel x86-64 (CISC) [1978~]

- レジスタ :

- RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, R8 ~ R15 → 64 ビット, 32ビット, 16ビット, 8ビットのアクセス



- RFLAGS: 状態レジスタ (overflow, sign, zero, carry, etc)
- Segment registers: DS, SS, CS ... → 最近はあまり使われない

- 命令語長 : 1 byte ~ 17 bytes

- 演算命令 : $A = A \text{ op } B$

- 出力オペランドは第一オペランドと同じ
- メモリオペランドも可能

“r”-operand: レジスタ

“r/m”-operand:
レジスタかメモリ

- 演算オペランドの種類 :

- 即値 (命令語に含まれる定数値)
- レジスタ
- メモリ: base-reg (+ index-reg*スケール値)(+ オフセット値)

x86-64 命令フォーマット

prefix	REX	opcode	modR/M	SIB	disp	imm
バイト数 (0~3)	(0~1)	(1~3)	(0~1)	(0~1)	(0~4)	(0~4)

- **prefix** (optional) : 各種モード設定
 - F0 : atomic instruction (exclusive mem R/W)
 - F2, F3 : Repeat instruction (string instr., IO instr.)
 - 2E, 36, 3E, 26, 64, 65 : segment override
 - 66, 67 : operand/address size override
- **REX** (opt) : 64ビットモード設定
- **opcode** (必須) : 1 ~ 3 bytes
- **modR/M** (opt) : オペランドモード (reg, mem addr)
 - reg-IDs: "r"-operand, "r/m" operand
- **SIB** (opt) : base-reg-ID, index-reg-ID, スケール値
 - **base-reg** + **index-reg** * **スケール値** + オフセット
- **disp** (opt) : アドレスオフセット値 (1, 2, 4 bytes)
- **imm** (opt) : 即値(定数) (1, 2, 4 bytes)

"r"-operand: レジスタ

"r/m"-operand:
レジスタかメモリ

x86-64 MOV 命令

(データ転送 : Load/Store)

88	+ 1~6B	: r/m8	← r8
89	+ 1~6B	: r/m32	← r32
8A	+ 1~6B	: r8	← r/m8
8B	+ 1~6B	: r32	← r/m32
B0	+ 1B	: r8	← imm8
B8	+ 4B	: r32	← imm32
C6	+ 2~7B	: r/m8	← imm8
C7	+ 5~10B	: r/m32	← imm32

第1バイト

追加バイト数

modR/M (1 byte)

mod	reg-ID	r/m-ID
2-bits	3-bits	3-bits

00 : r/m = M[reg]
01 : r/m = M[reg + disp8]
10 : r/m = M[reg + disp32]
11 : r/m = reg

- 第1バイト (opcode) : オペランドのビット長とタイプを決める
- imm8, imm32 : 8-bit/32-bit 即値
- r/m8, r/m32 : 8-bit/32-bit レジスタ・メモリオペランド
- r8, r32 : 8-bit/32-bit レジスタオペランド

x86-64 ADD 命令

04 + 1B	: AL	← AL + imm8
05 + 4B	: EAX	← EAX + imm32
81 + 5~10B	: r/m32	← r/m32 + imm32
83 + 2~7B	: r/m32	← r/m32 + imm8
00 + 1~6B	: r/m8	← r/m8 + r8
01 + 1~6B	: r/m32	← r/m32 + r32
02 + 1~6B	: r8	← r8 + r/m8
03 + 1~6B	: r32	← r32 + r/m32

第1バイト

追加バイト数

- 第1バイト (opcode) : オペランドのビット長とタイプを決める
- imm8, imm32 : 8-bit/32-bit 即値
- r/m8, r/m32 : 8-bit/32-bit レジスタ・メモリオペランド
- r8, r32 : 8-bit/32-bit レジスタオペランド

x86-64 Jcc 命令

(プログラム制御：条件分岐)

77 + 1B	: JA rel8 (if "above")
73 + 1B	: JAE rel8 (if "above or equal")
72 + 1B	: JB rel8 (if "below")
76 + 1B	: JBE rel8 (if "below or equal")
74 + 1B	: JE rel8 (if "equal")
75 + 1B	: JNE rel8 (if "not equal")
7F + 1B	: JG rel8 (if "greater")
7D + 1B	: JGE rel8 (if "greater or equal")
7C + 1B	: JL rel8 (if "less")
7E + 1B	: JLE rel8 (if "less or equal")

rel8 : 8-bit cur-PC
相対アドレス

"above", "below" :
符号なし大小比較

"greater", "less" :
符号付き大小比較

→ 同様の **Jcc** 命令群は**32-bit PC相対アドレス**でも存在する

- **RFLAGS レジスタ** : 状態を格納(比較結果など)
- **cur-PC** : 次の命令のアドレス値
- 分岐先アドレス = cur-PC + rel8

x86-64 コード例

013FA58770	40 55	push	rbp
013FA58772	57	push	rdi
013FA58773	48 81 EC D8 02 00 00	sub	rsp,2D8h
013FA5877A	48 8D 6C 24 30	lea	rbp,[rsp+30h]
013FA5877F	48 8B FC	mov	rdi,rbp
013FA58782	B9 B6 00 00 00	mov	ecx,0B6h
013FA58787	B8 CC CC CC CC	mov	eax,0CCCCCCCCh
013FA5878C	F3 AB	rep stos	dword ptr [rdi]
013FA5878E	C7 45 04 00 00 00 00	mov	dword ptr [rbp+04h],0
013FA58795	C7 45 24 10 00 00 00	mov	dword ptr [rbp+24h],10h
013FA5879C	C7 45 44 08 00 00 00	mov	dword ptr [rbp+44h],8
013FA587A3	B9 80 00 00 00	mov	ecx,80h
013FA587A8	E8 C9 8A FF FF	call	013FA51276h
013FA587AD	48 89 85 A8 01 00 00	mov	qword ptr [rbp+1A8h],rax
.....			

- 可変命令語長 → 命令フェッチ・命令デコード制御回路が複雑
- 多数の命令種 (~170種類) と多数のアドレスモード → コンパイラ設計が複雑

MIPS (RISC) [1985~]

- 背景：プロセッサのアーキテクチャを単純化することにより、処理速度の向上、回路の簡略化、コンパイラ設計の容易化を図る(プロセッサアーキテクチャの研究素材・教育素材としても普及)
- レジスタ：
 - 32 x 32-bit 汎用レジスタ
 - R0 : "zero" register(常に0の値を示す)
 - 専用 LO/HI registers (mult/div 結果保存用)
- 命令語長：32 bits
- 演算命令形式：A = B op C
 - 入力オペランド：レジスタ、即値
 - 出力オペランド(結果保存)：レジスタ
- メモリアクセス命令：load/store だけ

MIPS 命令形式

	6-bits	5-bits	5-bits	5-bits	5-bits	6-bits
Type-R	op	rs	rt	rd	sa	funct
Type-I	op	rs	rt	imm16		
Type-J	op	target26				

- **op** : 命令形式 (R/I/J) と命令種別を決定
- **rs, rt, rd** : reg-IDs
- **sa** : シフト量
- **funct** : Type-R形式の演算種別を決定
- **imm16** : 16-bit 即値
- **target26** : 26-bit 分岐先アドレス (PC相対)

MIPS 命令形式



$rd \leftarrow \text{funct}(rs, rt);$

$rd \leftarrow \text{funct}(rs, sa);$

funct :

100000(add), **100001**(addu), **100010**(sub), **100011**(subu)
100100(and), **100101**(or), **100110**(xor)
011010(div), **011011**(divu), **011000**(mult), **011001**(multu)
010000(move from HI : $rd \leftarrow HI$)
010010(move from LO : $rd \leftarrow LO$)
000000(shift-left : $rd \leftarrow rt \ll sa$)
000100(shift-left : $rd \leftarrow rt \ll rs$)
000011(shift right arithmetic : $rd \leftarrow rt \gg sa$)
000010(shift right logical : $rd \leftarrow rt \gg sa$)
101010(set on less than : $rd = (rs < rt)$)
101011(set on less than : $rd = (rs < rt) : \text{unsigned}$)
001000(jump register : $PC \leftarrow rs$)

add, sub : overflow例外あり

addu, subu : unsigned,
overflow 例外なし

MIPS 命令フォーマット



op :

[Compute]

001000(addi), **001001**(addiu)

$rt \leftarrow op(rs, imm16)$

001100(andi), **001101**(ori), **001110**(xori)

[Load]

100011(word), **100010**(half), **100101**(half unsigned),

100000(byte), **100100**(byte unsigned)

$rt \leftarrow M[rs + imm16]$

[Store]

101011(store word), **101001**(store half), **101000**(store byte)

[Branch]

000100(beq), **000101**(bneq)

$M[rs + imm16] \leftarrow rt$

if (rs op rt) $PC \leftarrow PC + imm16$

==, !=

MIPS 命令フォーマット



op :

000010(jump), **000011**(jump and link)

$PC \leftarrow (PC \& F0000000) \mid (\text{target26} \ll 2)$

$R31 \leftarrow PC + 8 \text{ (jump and link)}$

MIPS コード例

80000180	0001D821	addu \$27, \$0, \$1
80000184	3C019000	lui \$1, -28672
80000188	AC220200	sw \$2, 512(\$1)
8000018C	3C019000	lui \$1, -28672
80000190	AC240204	sw \$4, 516(\$1)
80000194	401A6800	mfc0 \$26, \$13
80000198	001A2082	srl \$4, 516(\$1)
8000019C	3084001F	andi \$4, \$4, 31
800001A0	34020004	ori \$2, \$0, 4
.....		

- 固定命令長 → 命令フェッチ・命令デコード制御回路が単純
- 少ない命令種 (~110種類) と限られたアドレスモード
- 演算オペランドはレジスタ → コンパイラ設計が比較的単純

まとめ

1. プロセッサ構成要素：メモリ、レジスタ、プログラムカウンタ, ALU, FPU
2. 命令実行制御
3. 論理回路による算術演算
4. 命令セット：CISC と RISC
 - 命令機能：データ転送、演算、プログラム制御
 - 命令フォーマット：機械語、アセンブリ言語、レジスタ転送記述
5. 命令セットの例：x86, MIPS

課題提出

1. $X = 19, Y = 30$ (10進表現) のとき、 $X + Y, X - Y, Y - X$ のそれぞれについて、2進表現での計算を資料P14の表記方法に従って示せ。ただし、入力と結果は6ビットとする。
2. $Z = X + Y$ を計算するプログラムを考える。ただし、 X と Y はメモリ上に格納されており、 Z の結果もメモリ上に格納するとし、 X, Y, Z とも32ビットデータとする。 $X86$ 命令セットの場合と $MIPS$ 命令セットの場合で、この計算を行うために必要な命令数と各命令の機能について説明せよ。
3. $CISC$ 命令セットと $RISC$ 命令セットの特徴を説明せよ。また、 $x86$ 命令セットに基づくプロセッサが現在もデスクトップPCやノートPCで主流となっている理由について、考察せよ。

- 提出期限：4/25(火) 17時
- 提出先：南3号館1階メールボックスS3-66(一色)