計算機論理設計
4. 命令形式と命令セット構成

一色 剛
工学院情報通信系
isshiki@ict.e.titech.ac.jp

アセンブリ言語と機械語

■ 機械語プログラム：

- 計算機が認識できる2値データによる命令表現

■ アセンブリ言語プログラム：

- テキスト形式の命令表現 → 機械語命令と等価

■ 命令形式（命令フォーマット）：

- 機械語の2値データと各命令のマッピングを定義
- 限られた命令ビット（本命令セットでは16ビット）に必要な命令情報（命令種別、アドレスモード、アドレス値）を埋め込む
- 命令解釈（命令デコード）のための論理回路をできるだけ簡単化するための符号（code）割当を考える

16ビットマイクロプロセッサ命令セット構成

[コンピュータアーキテクチャ, M.Morris Mano著, 1999]

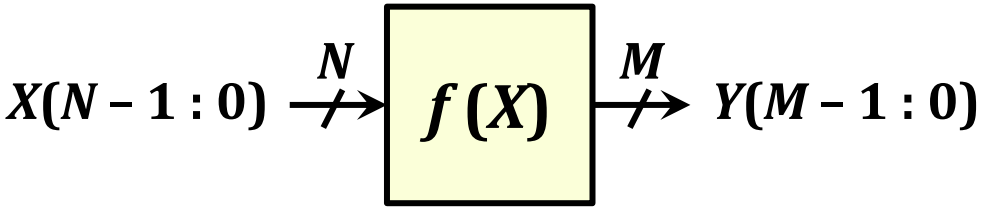
■ メモリ参照命令 :

- 命令中に12ビットアドレス値を含む
- アドレスモード : 直接アドレス、間接アドレス → 1ビットで表現
- 命令種別 : 残り3ビット(16 – 12 – 1 = 3)で表現
- メモリ参照命令（7個）: AND, ADD, LDA, STA, BUN, BSA, ISZ

■ 非メモリ参照命令:

- 命令中にアドレス値を含まず、他の付加情報も無い
- 命令種別 : 16ビットで表現可能 → ただし、「非メモリ参照命令」であることを表現する必要あり
- レジスタ参照命令（12個）: CLA, CLE, CMA, CME, CIR, CIL, INC, SPA, SNA, SZE, HLT
- 入出力命令（7個 → 詳細は後の講義で説明）: INP, OUT, SKI, SKO, ION, IOF, IMK

デコード回路と符号割当



デコード回路： 各出力ビットは特定の入力パターン（符号）のみ1（それ以外のパターンは0）

符号割当： 各出力ビットに対して値が1となる入力パターン（符号）を割当てる
→ 割当て方法によってデコーダ回路規模が異なる場合が出てくる

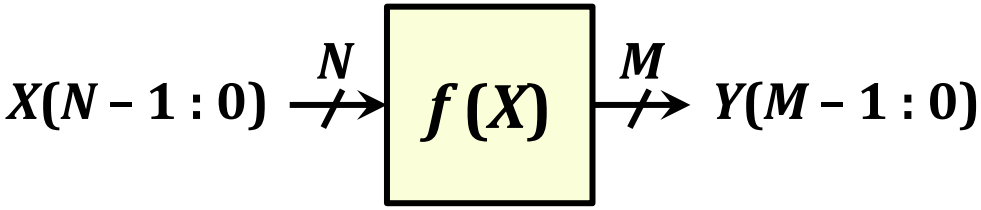
ここでの前提条件： 出力ビットが全て0になるような入力は**与えられない**とする
→ 出現しない入力パターンに対して出力値を「ドントケア」にできる

入力 $X(2:0)$	出力 $Y(7:0)$
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

入力 $X(2:0)$	出力 $Y(3:0)$
000	0001
001	0010
010	0100
011	1000
100	****
101	****
110	****
111	****

入力 $X(2:0)$	出力 $Y(2:0)$
000	001
001	010
010	100
011	***
100	***
101	***
110	***
111	***

デコード回路と符号割当



デコード回路： 各出力ビットは特定の入力パターン（符号）のみ1（それ以外のパターンは0）

$M = 2^N$の場合： N変数のAND項でデコーダ論理関数を表現できる
→ 異なる符号割当でも必要となるAND項の構成は同じ

入力 $X(2:0)$	出力 $Y(7:0)$	論理関数
000	00000001	$Y(0) = \overline{X(2)} \cdot \overline{X(1)} \cdot \overline{X(0)}$
001	00000010	$Y(1) = \overline{X(2)} \cdot \overline{X(1)} \cdot X(0)$
010	00000100	$Y(2) = \overline{X(2)} \cdot X(1) \cdot \overline{X(0)}$
011	00001000	$Y(3) = \overline{X(2)} \cdot X(1) \cdot X(0)$
100	00010000	$Y(4) = X(2) \cdot \overline{X(1)} \cdot \overline{X(0)}$
101	00100000	$Y(5) = X(2) \cdot \overline{X(1)} \cdot X(0)$
110	01000000	$Y(6) = X(2) \cdot X(1) \cdot \overline{X(0)}$
111	10000000	$Y(7) = X(2) \cdot X(1) \cdot X(0)$

入力 $X(2:0)$	出力 $Y(7:0)$	論理関数
000	00000001	$Y(0) = \overline{X(2)} \cdot \overline{X(1)} \cdot \overline{X(0)}$
001	00000100	$Y(2) = \overline{X(2)} \cdot \overline{X(1)} \cdot X(0)$
010	00010000	$Y(4) = \overline{X(2)} \cdot X(1) \cdot \overline{X(0)}$
011	01000000	$Y(6) = \overline{X(2)} \cdot X(1) \cdot X(0)$
100	00000010	$Y(1) = X(2) \cdot \overline{X(1)} \cdot \overline{X(0)}$
101	00001000	$Y(3) = X(2) \cdot \overline{X(1)} \cdot X(0)$
110	00100000	$Y(5) = X(2) \cdot X(1) \cdot \overline{X(0)}$
111	10000000	$Y(7) = X(2) \cdot X(1) \cdot X(0)$

N x Mデコーダ ($M = 2^N$)

デコード回路と符号割当

$M < 2^N$の場合：ドントケア出力が存在するため、異なる符号割当でAND項の形式が変わる → 回路規模が異なる

入力 $X(2:0)$	出力 $Y(3:0)$
000	0001
001	0010
010	0100
011	1000
100	* * * *
101	* * * *
110	* * * *
111	* * * *

$Y(0) = \overline{X(1)} \cdot \overline{X(0)}$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	1	0	0	0
1	*	*	*	*

$Y(1) = X(1) \cdot X(0)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	1	0	0
1	*	*	*	*

$Y(2) = X(1) \cdot X(0)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	0	0	1
1	*	*	*	*

$Y(3) = X(1) \cdot X(0)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	0	1	0
1	*	*	*	*

入力 $X(2:0)$	出力 $Y(3:0)$
000	0001
001	* * * *
010	0010
011	* * * *
100	* * * *
101	* * * *
110	0100
111	1000

$Y(0) = \overline{X(1)}$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	1	*	*	0
1	*	*	0	0

$Y(1) = X(2) \cdot X(1)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	*	*	1
1	*	*	0	0

$Y(2) = X(0)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	*	*	0
1	*	*	1	0

$Y(3) = X(2) \cdot X(0)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	*	*	0
1	*	*	0	1

デコード回路と符号割当

$M \leq N$の場合：入力の「1」の個数がちょうど1つの入力パターンに符号を割当てる（one-hot符号）→ 論理ゲートが不要になる

入力 $X(2:0)$	出力 $Y(2:0)$
000	001
001	010
010	100
011	* * *
100	* * *
101	* * *
110	* * *
111	* * *

$Y(0) = \overline{X(1)} \cdot \overline{X(0)}$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	1	0	*	0
1	*	*	*	*

$Y(1) = X(0)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	1	*	0
1	*	*	*	*

$Y(2) = X(1)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	0	0	*	1
1	*	*	*	*

one-hot符号

入力 $X(2:0)$	出力 $Y(2:0)$
000	* * *
001	001
010	010
011	* * *
100	100
101	* * *
110	* * *
111	* * *

$Y(0) = X(0)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	*	1	*	0
1	0	*	*	*

$Y(1) = X(1)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	*	0	*	1
1	0	*	*	*

$Y(2) = X(2)$

$X(2)$ \ $X(1:0)$	00	01	11	10
0	*	0	*	0
1	1	*	*	*

命令デコード回路設計

- **メモリ参照命令**
  - 命令中に12ビットアドレス値を含む
  - 命令種別 : 3ビットで表現
  - 7個のメモリ参照命令
- **非メモリ参照命令:**
  - 命令種別 : 16ビット表現可能
  - 12個のレジスタ参照命令
  - 7個の入出力命令
- **命令デコード回路設計**
  - 第1命令デコード回路 : 7個のメモリ参照命令種別と非メモリ参照命令の判定
  - 第2命令デコード回路 : 非メモリ参照命令の命令種別判定

16ビット命令形式

[コンピュータアーキテクチャ, M.Morris Mano著, 1999]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	OP1			ADI（アドレス値）											

メモリ参照命令
(OP1 ≠ 111)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	1	1	1	OP2（オプコード2）											

非メモリ参照命令
(OP1 = 111)

記号	ビット位置	説明
メモリ参照命令 (OP1 ≠ 111)		
I	(15)	0 : 直接アドレス，1 : 間接アドレス
OP1	(14 : 12)	オプコード1（命令コード）
ADI	(11 : 0)	アドレス値
非メモリ参照命令 (OP1 = 111)		
I	(15)	0 : レジスタ参照命令，1 : 入出力命令
OP2	(11 : 0)	オプコード2（命令コード）

第1命令デコード回路

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	OP1			ADI（アドレス値）											

メモリ参照命令
(OP1 ≠ 111)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	1	1	1	OP2（オプコード2）											

非メモリ参照命令
(OP1 = 111)

OP1(2:0)	命令	デコード論理関数
000	AND	$O_{AND} = \overline{OP1(2)} \cdot \overline{OP1(1)} \cdot \overline{OP1(0)}$
001	ADD	$O_{ADD} = \overline{OP1(2)} \cdot \overline{OP1(1)} \cdot OP1(0)$
010	LDA	$O_{LDA} = \overline{OP1(2)} \cdot OP1(1) \cdot \overline{OP1(0)}$
011	STA	$O_{STA} = \overline{OP1(2)} \cdot OP1(1) \cdot OP1(0)$
100	BUN	$O_{BUN} = OP1(2) \cdot \overline{OP1(1)} \cdot \overline{OP1(0)}$
101	BSA	$O_{BSA} = OP1(2) \cdot \overline{OP1(1)} \cdot OP1(0)$
110	ISZ	$O_{ISZ} = OP1(2) \cdot OP1(1) \cdot \overline{OP1(0)}$
111	非メモリ参照命令	$O_{NM} = OP1(2) \cdot OP1(1) \cdot OP1(0)$

第2命令デコード回路

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	1	1	1	OP2（オプコード2）											

非メモリ参照命令
(OP1 = 111)

レジスタ参照命令(I = 0)

OP2	命令
1000 0000 0000	CLA
0100 0000 0000	CLE
0010 0000 0000	CMA
0001 0000 0000	CME
0000 1000 0000	CIR
0000 0100 0000	CIL
0000 0010 0000	INC
0000 0001 0000	SPA
0000 0000 1000	SNA
0000 0000 0100	SZA
0000 0000 0010	SZE
0000 0000 0001	HLT

入出力命令(I = 1)

OP2	命令
1000 0000 0000	INP
0100 0000 0000	OUT
0010 0000 0000	SKI
0001 0000 0000	SKO
0000 1000 0000	ION
0000 0100 0000	IOF
0000 0010 0000	IMK

one-hot符号

第2命令デコード回路

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	1	1	1	OP2（オプコード2）											

非メモリ参照命令
(OP1 = 111)

レジスタ参照命令(I = 0)

OP2	命令	デコード論理関数
1000 0000 0000	CLA	$O_{CLA} = O_{NM} \cdot \bar{I} \cdot OP2(11)$
0100 0000 0000	CLE	$O_{CLE} = O_{NM} \cdot \bar{I} \cdot OP2(10)$
0010 0000 0000	CMA	$O_{CMA} = O_{NM} \cdot \bar{I} \cdot OP2(9)$
0001 0000 0000	CME	$O_{CME} = O_{NM} \cdot \bar{I} \cdot OP2(8)$
0000 1000 0000	CIR	$O_{CIR} = O_{NM} \cdot \bar{I} \cdot OP2(7)$
0000 0100 0000	CIL	$O_{CIL} = O_{NM} \cdot \bar{I} \cdot OP2(6)$
0000 0010 0000	INC	$O_{INC} = O_{NM} \cdot \bar{I} \cdot OP2(5)$
0000 0001 0000	SPA	$O_{SPA} = O_{NM} \cdot \bar{I} \cdot OP2(4)$
0000 0000 1000	SNA	$O_{SNA} = O_{NM} \cdot \bar{I} \cdot OP2(3)$
0000 0000 0100	SZA	$O_{SZA} = O_{NM} \cdot \bar{I} \cdot OP2(2)$
0000 0000 0010	SZE	$O_{SZE} = O_{NM} \cdot \bar{I} \cdot OP2(1)$
0000 0000 0001	HLT	$O_{HLT} = O_{NM} \cdot \bar{I} \cdot OP2(0)$

第2命令デコード回路

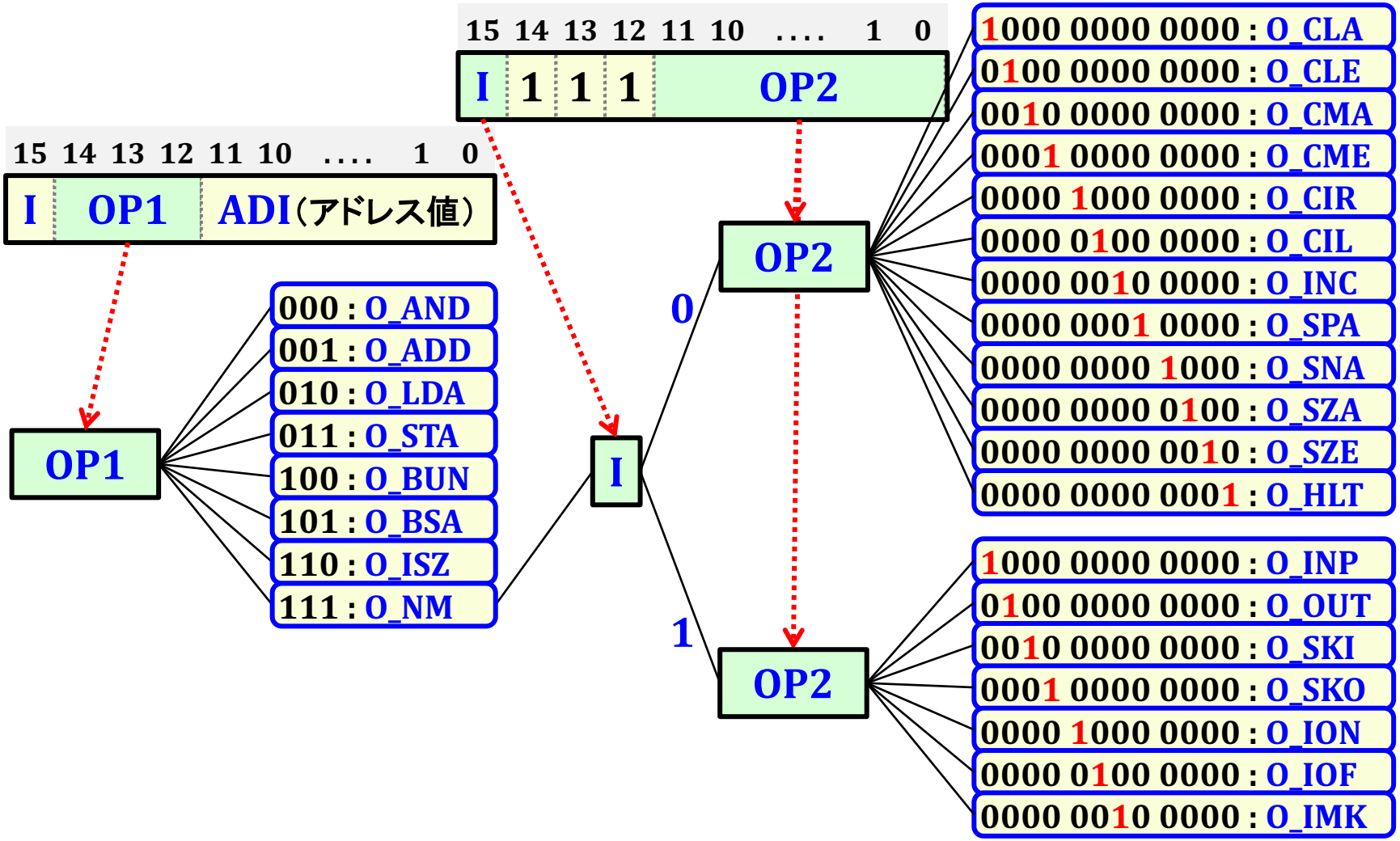
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I	1	1	1	OP2（オプコード2）											

非メモリ参照命令
(OP1 = 111)

入出力命令(I = 1)

OP2	命令	デコード論理関数
1000 0000 0000	INP	$O_{INP} = O_{NM} \cdot I \cdot OP2(11)$
0100 0000 0000	OUT	$O_{CLE} = O_{NM} \cdot I \cdot OP2(10)$
0010 0000 0000	SKI	$O_{CMA} = O_{NM} \cdot I \cdot OP2(9)$
0001 0000 0000	SKO	$O_{CME} = O_{NM} \cdot I \cdot OP2(8)$
0000 1000 0000	ION	$O_{CIR} = O_{NM} \cdot I \cdot OP2(7)$
0000 0100 0000	IOF	$O_{CIL} = O_{NM} \cdot I \cdot OP2(6)$
0000 0010 0000	IMK	$O_{INC} = O_{NM} \cdot I \cdot OP2(5)$

命令デコード木


15 14 13 12 11 10 .... 1 0
I 1 1 1 OP2
15 14 13 12 11 10 .... 1 0
I OP1 ADI（アドレス値）
OP1
000 : O_AND
001 : O_ADD
010 : O_LDA
011 : O_STA
100 : O_BUN
101 : O_BSA
110 : O_ISZ
111 : O_NM
I
0
1
OP2
1000 0000 0000 : O_CLA
0100 0000 0000 : O_CLE
0010 0000 0000 : O_CMA
0001 0000 0000 : O_CME
0000 1000 0000 : O_CIR
0000 0100 0000 : O_CIL
0000 0010 0000 : O_INC
0000 0001 0000 : O_SPA
0000 0000 1000 : O_SNA
0000 0000 0100 : O_SZA
0000 0000 0010 : O_SZE
0000 0000 0001 : O_HLT
OP2
1000 0000 0000 : O_INP
0100 0000 0000 : O_OUT
0010 0000 0000 : O_SKI
0001 0000 0000 : O_SKO
0000 1000 0000 : O_ION
0000 0100 0000 : O_IOF
0000 0010 0000 : O_IMK

命令セットの基本要件

- 命令セットとして以下の基本機能が実現可能なこと：
  - 演算機能：四則演算、論理演算、シフト演算
  - メモリアクセス機能：ロード、ストア
  - プログラム制御機能：分岐（無条件・条件付き）、サブルーチン（呼出し・復帰）
- 本16ビット命令セット：基本要件を「一応」満たしている
  - 演算機能：AND, ADD, CMA, CIL, CIRの組合せで、全ての四則演算、論理演算、シフト演算は実現可能
  - メモリアクセス機能：LDA, STA, ISZ
  - プログラム制御機能：
    - ✓ 無条件分岐（BUN）、条件分岐（条件スキップ命令 + BUN）
    - ✓ サブルーチン呼出し（BSA）、復帰（BUN 間接アドレス）

→ 計算機としての性能改善の余地は多い：

- ✓ レジスタ数・演算命令数の増強
- ✓ 即値データへの対応：定数、アドレスオフセット
- ✓ コード密度：one-hot符号は回路規模は小さいが、冗長な情報

命令セットの評価指標

- コード密度：高コード密度 → 小プログラムメモリ
  - ある機能を実現するための機械語命令列のサイズ
  - 命令の高機能化・複雑化や可変長命令によってコード密度は向上する

  → コード「品質」に大きく依存（コンパイラ、アセンブリプログラミング）
- 実行時間= 実行サイクル数×サイクル周期
  - 実行サイクル数 → 1命令の実行サイクル数に依存
  - サイクル周期 → 組合せ回路の最大論理遅延に依存

  → 命令実行制御を実現するハードウェアアーキテクチャに依存
- コンパイラ開発容易性：
  - コンパイラの「性能」→ 実行命令総数、コード密度
  - 複雑な機能を持つ命令セット用のコンパイラ開発が難しい
- 命令セット設計の難しさ：（命令セットの寿命が長い理由）
  - 計算機システム全体の完成まで、命令セットの「良し悪し」が評価しにくい
  - 命令セットの小さな修正が計算機システム設計全体に波及する

命令セット分類 : CISC と RISC

- **CISC : Complex Instruction-Set Computer (Intel x86など)**
  - 背景 : 初期の計算機、アセンブリプログラミング中心のSW開発
  - メモリ参照演算主体の命令体系（命令の高機能化）、可変長命令→ 高コード密度
  - レジスタ数は少ない（当時はレジスタが高価、アセンブリプログラミングでは多数のレジスタを使いこなすことが困難）
  - 命令実行制御が複雑 → micro-code制御方式（命令実行制御回路をプログラム可能な順序回路で実現）
- **RISC : Reduced Instruction-Set Computer (MIPS, ARMなど)**
  - 背景 : SW規模増大、高級言語の必要性 → コンパイラ開発を優先した命令体系の単純化（Reduced Instruction-Set）
  - メモリ参照は、ロード・ストア命令のみ
  - 「汎用レジスタ」を多数搭載することで、冗長なメモリ参照を軽減（プログラム変数のレジスタ割当問題は、コンパイラの最適化機能の一つ）
  - 固定長命令、論理回路構造の単純化、パイプライン型命令実行制御
- **本16ビット命令セット → CISCに分類される**
  - メモリ参照演算あり、レジスタはACのみ

レジスタ構成例

- (CISC) x86レジスタ構成
  - EAX (accumulator reg.), EBX (base reg.), ECX (count reg.), EDX (data reg.), EDI (destination index), ESI (source index), ESP (stack pointer), EBP (base pointer)
  - 全て汎用レジスタとして使えるが、幾つかの命令では専用の役割
  - 現在の64ビットマシンでは、上記8つ以外に8つ新たな汎用レジスタを追加（x86データ幅の変遷 : 16ビット → 32ビット → 64ビット）
- (RISC) MIPSレジスタ構成
  - 32汎用レジスタ（32ビット） → 全ての命令で参照可能
  - R0レジスタ = 「0」レジスタ → 常に0を出力、書込みは無効
- (RISC) ARMレジスタ構成
  - 16汎用レジスタ（32ビット） → 全ての命令で参照可能
  - PC (program counter)も汎用レジスタの一つ → 多様な分岐命令の実現方法やデータメモリ参照方法が可能
- (CISC) 本16ビット命令セットレジスタ構成 : ACのみ

命令形式（メモリ参照・演算）

■ x86命令形式：可変長命令（1バイト ～ 17バイト）

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

- 演算：A ← A op B（A, Bはメモリオペランドも可）

■ MIPS命令形式：32ビット固定長命令

op(6)	rs(5)	rt(5)	rd(5)	sa(5)	funct(6)
op(6)	rs(5)	rt(5)	imm(16)		

- 演算：A ← B op C（レジスタのみ、op/functで演算指定）

■ ARM命令形式：32ビット固定長命令（16ビット命令切替可）

cond(4)	01	mode(6)	Rn(4)	Rd(4)	operand2(12)
cond(4)	00	mode(6)	Rn(4)	Rd(4)	offset(12)

- 演算：A ← B op C (レジスタのみ、modeで演算指定）
- 全ての命令は「条件付き実行命令」（condで条件指定）

■ 本16ビットプロセッサ命令形式：16ビット固定長命令

15	14 13 12	11 10 .... 1 0
I	OP1	ADI / OP2

レジスタ指定方法

- x86 : modR/M（1バイト）でレジスタを指定

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

  - opcodeにレジスタ指定する命令や固定レジスタ命令もある

- MIPS : 5ビットレジスタフィールド（rs, rt, rd）で指定

op(6)	rs(5)	rt(5)	rd(5)	sa(5)	funct(6)	(1)
op(6)	rs(5)	rt(5)	imm(16)			(2)

  - 演算命令 : rd ← rs op rt (1), rt ← rs op imm (2)
  - ロード・ストア (2) : rt（データ）、rs + imm（アドレス値）

- ARM : 4ビットレジスタフィールド（Rn, Rd）で指定

cond(4)	01	mode(6)	Rn(4)	Rd(4)	operand2(12)	演算命令
cond(4)	00	mode(6)	Rn(4)	Rd(4)	offset(12)	ロード・ストア

  - 演算命令 : Rd ← Rn op (operand2)
  - ロード・ストア : Rd（データ）、Rn + offset（アドレス値）

- 本16ビットプロセッサ : レジスタは各命令で固定

即値（定数）データ指定方法

- x86 : disp/immフィールド（1バイト or 4バイト）で指定

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

  - アドレス値 : ベースアドレス + disp値（2の補数）
  - ベースアドレス : base-reg + idx-reg * scaleも可 (SIB拡張)
- MIPS : 16-bit即値（2の補数）

op(6)	rs(5)	rt(5)	imm(16)

  - 即値オペランド : rt ← rs op imm
  - アドレス値 : rs + imm
- ARM : 8-bit即値、12-bitアドレスオフセット（符号なし）

cond(4)	01	mode(6)	Rn(4)	Rd(4)	operand2(12)	演算命令
cond(4)	00	mode(6)	Rn(4)	Rd(4)	offset(12)	ロード・ストア

  - operand2の即値指定 : 8-bit即値 + 回転シフト操作
  - offsetのアドレスオフセット指定 : 12-bit（modeで±指定）
- 本16ビットプロセッサ : 即値は指定できない

実行条件指定方法（x86, ARM）

- 状態フラグレジスタによる条件指定
  - x86 (EFLAGS) : ZF(zero), CF(carry), SF(sign), OF(overflow)等
  - ARM (CPSR): Z(zero), C(carry), N(sign), V(overflow)
- 状態フラグレジスタを変更する命令（条件設定命令）
  - x86 : TEST, CMP（比較演算）、他の演算命令（ADD, SUB等）
  - ARM ：TST, TEQ, CMP, CMN（比較演算）、他命令
- CMP A, B → D = A – B の結果に基づいて状態フラグを設定
  - D : K + 1ビット
  - A, B : Kビット（符号なし）
  - ZF = (D(K – 1:0) = 0)
  - CF = D(K), SF = D(K – 1)
  - OF = (A(K – 1) ⊕ B(K – 1)) & (A(K – 1) ⊕ SF)

等価比較 :
A == B : ZF
A != B : $\overline{ZF}$

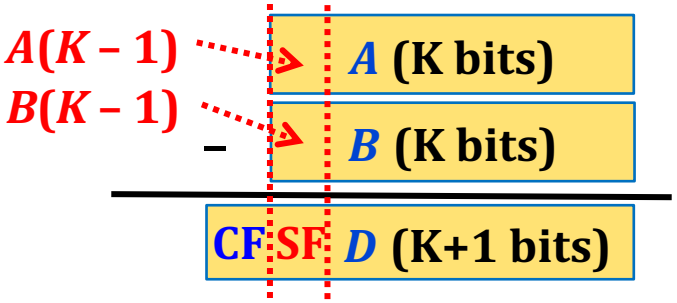
符号なし比較 :
A >= B : $\overline{CF}$
A < B : CF
A > B : $\overline{CF} \cdot \overline{ZF}$
A <=B : $CF + ZF$

2の補数比較 :
A >= B : $SF = OF$
A < B : $SF \neq OF$
A > B : $(SF = OF) \cdot \overline{ZF}$
A <= B :$(SF \neq OF) + ZF$

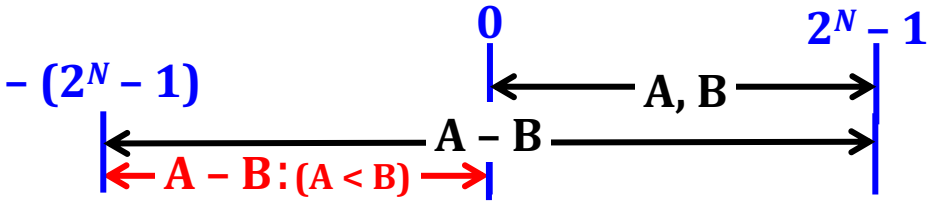
OF : 2の補数オーバーフロー
AとBを2の補数とした時、D = A – BがKビットの2の補数で表現できない場合に「オーバーフロー」が起きる

比較演算（符号なし・2の補数）

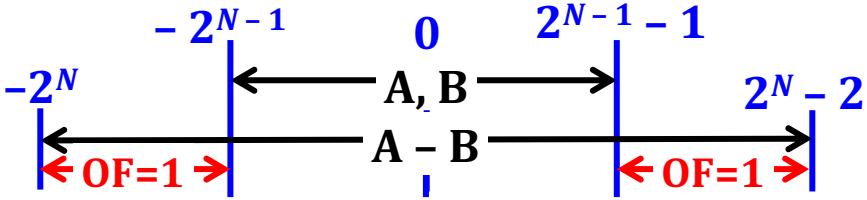
- **D = A – B**
  - D : K + 1ビット
  - A, B : Kビット
  - ZF = (D(K – 1:0) = 0)
  - CF = D(K), SF = D(K – 1)
  - OF = (A(K – 1) ⊕ B(K – 1)) & (A(K – 1) ⊕ SF)



- **符号なしデータの比較**
  - A, B : Kビット符号なし
  - D : K + 1ビット2の補数



- **2の補数データの比較**
  - A, B, D : Kビット2の補数
  - オーバーフロー区間（OF = 1）では、Dの符号（SF）は反転している
  - オーバーフロー条件 : (AとBの符号が異なる) & (AとDの符号が異なる)



比較演算

D = A – B
D : N + 1ビット、A, B : Nビット（符号なし）
ZF = (D(K – 1:0) = 0) ,
CF = D(K) , SF = D(K – 1) ,
OF = (A(K – 1) ⊕ B(K – 1)) & (A(K – 1) ⊕ SF)

	unsigned	signed
11111000	248	–8
10001000	136	–120
00010000	16	16

11111000	+248	–8
– 10001000	–136	+120
001110000	+112	+112
OF = (1⊕1) & (1⊕0)=0		
(ZF,CF,SF,OF) = (0,0,0,0)		
248 > 136, (–8) > (–120)		

11111000	+248	–8
– 00010010	–18	–18
011100110	+230	–26
OF = (1⊕0) & (1⊕1)=0		
(ZF,CF,SF,OF) = (0,0,1,0)		
248 > 18, (–8) < (18)		

10001000	+136	–120
– 00010010	–18	–18
001110110	+118	–138
OF = (1⊕0) & (1⊕0)=1		
(ZF,CF,SF,OF) = (0,0,0,1)		
136 > 18, (–120) < (18)		

10001000	+136	–120
– 11111000	–248	+8
110010000	–112	–112
OF = (1⊕1) & (1⊕1)=0		
(ZF,CF,SF,OF) = (0,1,1,0)		
136 < 248, (–120) < (–8)		

00010010	+18	+18
– 11111000	–248	+8
100011010	–230	+26
OF = (0⊕1) & (0⊕0)=0		
(ZF,CF,SF,OF) = (0,1,0,0)		
18 < 248, (18) > (–8)		

00010010	+18	+18
– 10001000	–136	+120
110000110	–112	+138
OF = (0⊕1) & (0⊕1)=1		
(ZF,CF,SF,OF) = (0,1,1,1)		
18 < 136, (18) > (–120)		

比較演算

等価比較：
A == B : ZF
A != B : $\overline{ZF}$

符号なし比較：
A >= B : $\overline{CF}$
A < B : CF
A > B : $\overline{CF} \cdot \overline{ZF}$
A <=B : $CF + ZF$

2の補数比較：
A >= B : $SF = OF$
A < B : $SF \neq OF$
A > B : $(SF = OF) \cdot \overline{ZF}$
A <= B :$(SF \neq OF) + ZF$

11111000	+248	−8
− 10001000	−136	+120
001110000	+112	+112
OF = (1⊕1) & (1⊕0)=0		
(ZF,CF,SF,OF) = (0,0,0,0)		
248 > 136, (−8) > (−120)		

11111000	+248	−8
− 00010010	−18	−18
011100110	+230	−26
OF = (1⊕0) & (1⊕1)=0		
(ZF,CF,SF,OF) = (0,0,1,0)		
248 > 18, (−8) < (18)		

10001000	+136	−120
− 00010010	−18	−18
001110110	+118	−138
OF = (1⊕0) & (1⊕0)=1		
(ZF,CF,SF,OF) = (0,0,0,1)		
136 > 18, (−120) < (18)		

10001000	+136	−120
− 11111000	−248	+8
110010000	−112	−112
OF = (1⊕1) & (1⊕1)=0		
(ZF,CF,SF,OF) = (0,1,1,0)		
136 > 248, (−120) > (−8)		

00010010	+18	+18
− 11111000	−248	+8
100011010	−230	+26
OF = (0⊕1) & (0⊕0)=0		
(ZF,CF,SF,OF) = (0,1,0,0)		
18 < 248, (18) > (−8)		

00010010	+18	+18
− 10001000	−136	+120
110000110	−112	+138
OF = (0⊕1) & (0⊕1)=1		
(ZF,CF,SF,OF) = (0,1,1,1)		
18 < 136, (18) > (−120)		

条件分岐・条件実行

- 複数の状態フラグレジスタによる条件指定
  - x86 (EFLAGS) : ZF(zero), CF(carry), SF(sign), OF(overflow)等
  - ARM (CPSR): Z(zero), C(carry), N(sign), V(overflow)
- 状態フラグレジスタにおける条件実行
  - x86 : Jcc (jump on condition), SETcc (set byte on condition)
  - ARM ：全命令
- MIPSの条件分岐
  - 状態フラグは持たず、2つのレジスタの等価比較(==, !=)による条件分岐だけ
- 本16ビットプロセッサの条件分岐
  - 条件設定命令: ISZ, SPA, SNA, SZA, SZE
  - 条件スキップ命令のみ

本16ビット命令セットの改善ポイント

■ P15 : 計算機としての改善の余地

- ✓ レジスタ数・演算命令数の増強
- ✓ 即値データへの対応 : 定数、アドレスオフセット
- ✓ コード密度 : one-hot符号は回路規模は小さいが、冗長な情報

① 可変長命令化

- 例1）非メモリ参照命令を1バイト命令にする → 高コード密度
- 例2）メモリ参照命令を3バイト（命令1バイト + アドレス2バイト）命令にする
- 例3）即値データを命令の後続バイトで指定

② 演算命令数増強

- 例2により、メモリ参照命令（2項演算命令）を大幅に増強可能

③ レジスタ数増強

- 多数のレジスタを増強する場合は、命令1バイトでは足りなくなる恐れがあるので、命令情報だけで2バイトにする必要が出てくる

→ 3年次1Q「情報通信実験3」でチャレンジしましょう！