計算機工学

第２章

基本アーキテクチャ

教科書 コンピュータアーキテクチャの基礎，柴山潔先生著(京都工芸繊維大学)

参考書 コンピュータの構成と設計，パターソン&ヘネシー

ノイマン型コンピュータ

ほとんどすべての計算機のとる基本的なアーキテクチャ

フォンノイマン ENIACを開発

２進数演算 ０と１の２値で命令とデータを表現

プログラム内蔵 あらかじめ，プログラムをメモリに格納しておき，メモリの格納順に実行する.

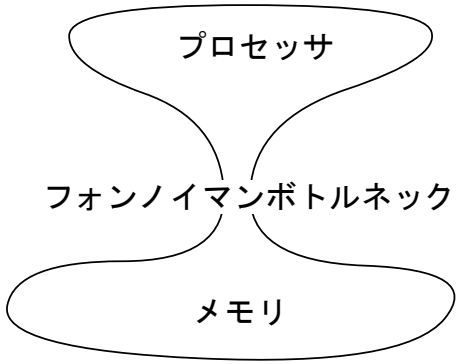
プロセッサとメモリ

プロセッサ 命令列にしたがって，処理を実行

メモリ 内蔵プログラム，データを格納

PC プログラムカウンタ．通常は命令列を順に実行

フォンノイマンボトルネック メモリとのデータのやり取りが遅くて，プロセッサの処理能力が制限される．



教科書 P31 図2.1より

フォンノイマンボトルネックの実体

メモリの速度(DDR SDRAM)

規格	クロック周波数	データ幅	帯域幅
PC2100	133MHz×2倍	64bit	2.13Gbytes/s
PC2700	166MHz×2倍	64bit	2.66Gbytes/s
PC3200	200MHz×2倍	64bit	3.20Gbytes/s

プロセッサの処理能力

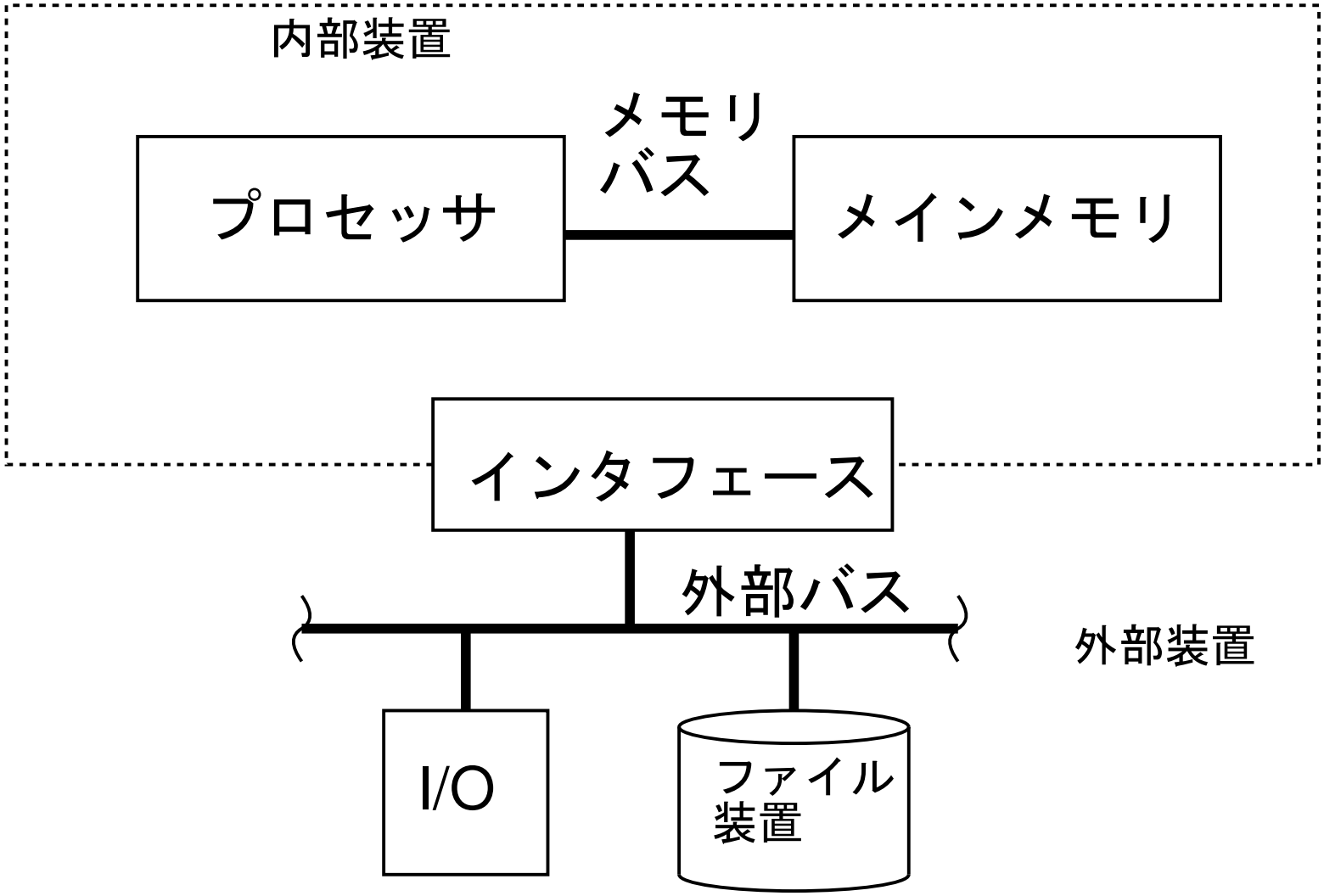
- Pentium4(P4)は, 整数演算器x2(クロック周波数の倍で動作)
- 最大整数演算能力は, 4byte×2×クロック周波数
- 2GHzなら, 16Gbytes/s

実際の処理能力

	AthlonXP2000+	P4 2.2GHz	P4 2GHz
Dhrystone(MIPS)	4714	4279	3860
Whetstone(MFLOPS)	2323	2685	2443

参考: http://pcweb.mycom.co.jp/articles/2002/01/16/02/bench/002.html

コンピュータの構造



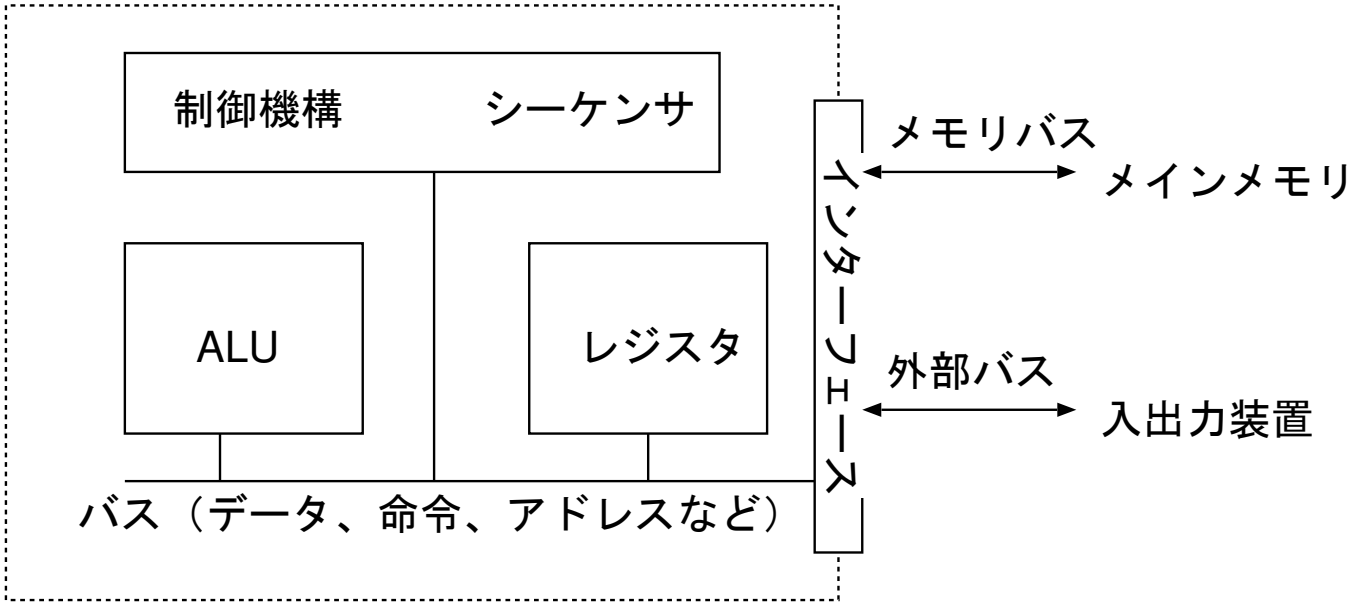
教科書P31 図2.2より

プロセッサの内部構造

レジスタ プロセッサ内部でデータを格納．プロセッサの１クロックサイクルで読み書き可能．レジスタファイル: レジスタの集合

ALU Arithmetic and Logic Unit． レジスタやメモリの値の演算処理を実行する．

制御機構 演算結果等により，プロセッサの動作を制御する．



教科書P34 図2.3より

メインメモリ

プログラムとデータを格納．通常DRAM (Dynamic RAM)で実現.

入出力装置

I(nput)/O(utput)装置

例をあげてみよう

ファイル装置

ハードディスク．通常，OSが制御する．メインメモリにロードするためのデータとプログラムを格納.

バスとインターフェース

バス 装置間を結ぶ配線．同じ配線上に複数の装置がつながっている場合が多い．

インターフェース バス間をつなぐ境界

内部バス プロセッサ構造の内部にあるバス．

外部バス 計算機を拡張するためのバス．

プロセッサ内部のバス データバス，命令バス，アドレスバス．

基本命令セットアーキテクチャ

Instruction Set Architecture (ISA)

命令セット プロセッサで実行される命令の一覧．計算機に読み込まれるデータはすべて，01の２進数．ISAにより，その２進数列に意味が与えられる．

- プロセッサの違いは，ほとんどの場合ISAの違い．ISAが同じでも，内部構造(マイクロアーキテクチャ)は，異なる場合もある．

容量の単位

ビット(bit) ２進数一つ

バイト(Byte, B) ８ビットの２進数

ワード(Word, W) 計算機で一度に取り扱えるビット数．通常32ビット

C言語とn進数

a=0x415; 16進数(4ビット)による表現．1Byte: 2桁
a=04152; 8進数(3ビット)による表現
bool a; 真(TRUE, 1)か, 偽(FALSE, 0)
char a; 8bit
short int b; 16bit
long int c; 32bit(1ワード)

アナログとディジタル

アナログ 時間的に連続な量

ディジタル 時間的，量的に不連続な量で表すこと．

標本化 サンプリング．時間的に離散化する．

量子化 サンプリングして得られた量を離散化する．

A/D 変換，D/A 変換 アナログ信号，ディジタル信号の相互の変換．

2 進数 計算機内部では，ディジタル化した情報を，2 進数で表現．

マイクロプロセッサのビット幅

Intel i4004	4ビット
i8086	8ビット
i80286	16ビット
i80386	32ビット
Intel Xeon(P4) AMD Opteron AMD Athlon64	一部64ビット拡張
Intel IA-64(Itanium)	64ビット

- nビットのプロセッサは, nビットまでのデータ処理が得意. それ以上のビット幅の処理も一部可能であるが, 性能が落ちる.
- 4bitで処理するという画期的なアイディアで一つのLSIに集積化されたマイクロプロセッサも, 集積回路技術の進歩により, その処理能力は飛躍的に向上.

命令とデータ

命令とデータのアドレスを格納しているレジスタ

PC (Program Counter) 実行しているプログラムの命令のアドレスを格納

MAR (Memory Address Register) データとしてアクセスしているメモリのアドレスを格納.

プログラム

命令列だけでは, 何もできない. データはプログラム中, もしくは外部データとして提供される.

命令形式

- 命令＝命令そのものと，命令に与えるデータを指定する部分からなる．２進数のデータ
- アセンブラは，人間が理解しやすいように，命令列を文字列に変換したもの
- C言語をコンパイルしてできるバイナリコード中に格納されている．

命令コード operation code，命令そのもの．add，sub，xor，load，store

オペランド operand，命令に与えるデータを指定．データそのもの，レジスタのアドレス，メモリのアドレス．

ソースオペランド 命令実行によって処理される側

デスティネーションオペランド 命令実行によって生ずる結果を格納する側

オペランドの指定対象

メインメモリ メモリはサイズが大きいので，アドレスを指定するのに多ビット必要．

レジスタ 高速に複数のレジスタに同時にアクセス可能．通常は，２読出１書込．レジスタの数が増えると，オペランドのビット幅が増える．

代表的なプロセッサのレジスタ数

x86	８個
MIPS	16個 (R0は0レジスタ)
SPARC	32個
PowerPC	32個

レジスタの種類

汎用レジスタ 様々な用途に使えるレジスタ．プロセッサによっては，特定のアドレスのレジスタを特殊用途に使う場合もある．MIPSのR0 は0レジスタ

専用レジスタ 汎用レジスタとは別に存在する特殊用途のレジスタ．浮動小数点レジスタ，インデックスレジスタ (後述)

命令形式

指定できるオペランドの数により分類.

3アドレス DST×1, SRC×2

2アドレス (DST兼SRC)×1, SRC×1

1アドレス SRC×1, 専用レジスタ(accumulator, AC, DST兼SRC)×1. アキュムレータマシン

0アドレス スタックに詰んでいく. JAVA言語で規定している仮想計算機は, スタックのみを使用できる. スタックマシン.

- オペランドがメモリ(M), レジスタ(R). R-R-R,M-M, R-M, M-R, R-R
- 次命令は, メモリの次のアドレス(ワード単位). 条件分岐等を除く

主な命令の種類

分類	略称	実行内容
データ転送	LOAD	メモリ内容をレジスタに転送
	STORE	レジスタの内容をメモリに転送
	MOV	レジスタ間で値を転送
整数演算	ADD, SUB	レジスタ/メモリ間の加算
	MUL, DIV	乗除算
	COMP	比較して結果を特殊レジスタに格納
論理演算	OR, AND, XOR	論理和，論理積，排他的論理和
	SH[RL]	1ビット論理 [右左] シフト
	SA[RL]	1ビット算術 [右左] シフト
	RO[RL]	1ビット循環 [右左] シフト
制御	JMP	無条件分岐
	JPcc	ccで指定した条件が真なら分岐
	INT	割り込み発生
	CALL	サブルーチンコール
	RET	サブルーチンコールからの復帰

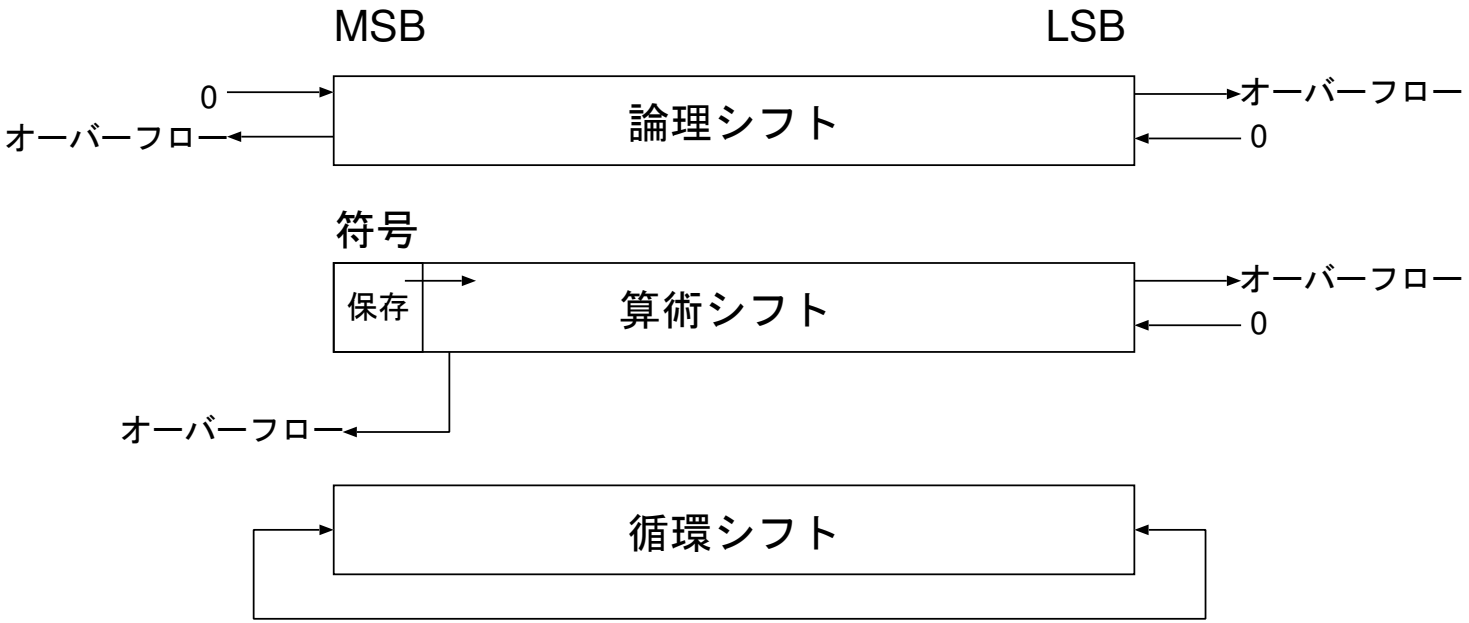
シフト命令

ビット列を左右にずらす.

論理シフト そのまま何も考えずに, 左右にシフトする. 左右からは0が入力される.

算術シフト 符号ビット(MSB)を保ったまま左右にシフトする. 左シフト=2倍, 右シフト=1/2倍. 右シフトの場合は, 符号ビットがそのままシフトする.

循環シフト MSBからあふれたビットはLSBに, LSBからあふれたビットはMSBに循環する.



教科書P61 図2.31より

シフトの実例(100, −100を右に3ビットシフト)

	2進数表記								符号付の10進数表記
元の数	0	1	1	0	0	1	0	0	100 (64+32+4)
算術シフト									
論理シフト									
循環シフト									

	2進数表記								符号付の10進数表記
元の数									−100
算術シフト									
論理シフト									
循環シフト									

CISCとRISC

細かい定義はいろいろあるが，命令形式で言うと，

CISC Complex Instruction-Set Computer，メモリを直接演算のオペランドとして使用できる．

RISC Reduced Instruction-Set Computer，演算のオペランドとして使用できるのは，レジスタのみ．

RISC型計算機では，ハードウエアが簡単になり，クロック周波数があげやすく，処理速度が上がる．CISC型計算機は，メモリ/レジスタのハードウエア単価が高かった時代のアーキテクチャ(RISCはより多くのメモリとレジスタが必要)．

アドレス指定モード

アドレッシング メモリのアドレスの付け方．アドレス空間

メモリアドレス 32 ビットアドレス： $2^{32} = 4\text{GByte}$，24 ビットアドレス:$2^{24} =$ 16MByte 16 ビットなら $2^{16} =$

- 命令に絶対アドレスを直接埋め込むことは少ない．

アドレス指定モード(2)

命令のオペランドから，実効アドレスを生成.

絶対アドレス 1オペランドで直接メモリのアドレスを指定

相対アドレス アドレス＝上位ビット＋下位ビット.

データの読出し方法 EA＝102に入った，312というデータにアクセス.

- **直接アドレス** 312＝Mem[EA]，EA＝opr
- **間接アドレス** 312＝Mem[Mem[opr]]，EA＝Mem[opr]．**C言語のポインタ!**

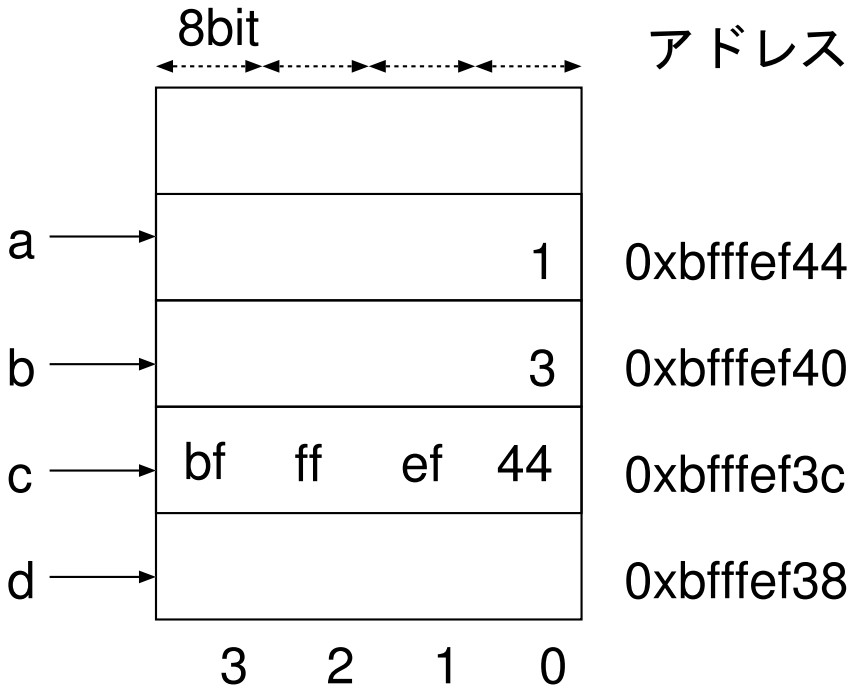
即値 オペランド＝データ．例：c＝a＋5;

C言語で実践

```
main(){
  int a=1,b=3;
  int *c,*d; //cは, int型へのポインタ
  c=&a; //cにaのアドレスを代入
  printf("a = %d, b= %d\n",a, b); //a, bの中身を表示
  printf("c = %x\n",c); //cの中身を表示
  printf("*c= %d\n",*c);  //cに格納されているアドレスの中身を表示
  printf("&a= %x\n",&a); //aのアドレスを表示
  printf("&b= %x\n",&b); //bのアドレスを表示
  printf("&c= %x\n",&c); //cのアドレスを表示
  printf("&d= %x\n",&d); //dのアドレスを表示
}
```

実行結果

マシン１	マシン２
a = 1, b= 3	a = 1, b= 3
c = bfffef44	c = efffe4a4
*c= 1	*c= 1
&a= bfffef44	&a= efffe4a4
&b= bfffef40	&b= efffe4a0
&c= bfffef3c	&c= efffe49c
&d= bfffef38	&d= efffe498



ポインタ変数自体は，アドレスを保存している．　メインメモリの中身

*c	cが保持するアドレスが指す先の中身
c	cの内容
&c	cのアドレス

各種のアドレス指定モード

具体的な指定方法(C言語では見えない).

レジスタ直接 レジスタの中身に, データが格納.

レジスタ間接 レジスタの中身に, データの入っているメインメモリのアドレス

インデックス インデックスレジスタ+即値(固定). インデックスレジスタは, 下位アドレス. データの順次アクセス

ベース ベースレジスタ+即値(固定), ベースレジスタは上位アドレス. メモリ空間の移動

ベースインデックス ベース+インデックス

PC相対 PC(プログラムカウンタ)+即値. 条件分岐.

基本データ型

すべて２進数で表現される．人間が見るときの型

数値 **整数 (integer)** Ｃでは，int．有限範囲の小数も表すことができる (固定小数点)

実数 (real) ＣではFloat, double．無限範囲の数を表現することができる．float とdoubleは精度が異なる．(浮動小数点)

論理値 (boolean) Ｃでは，bool．真か偽か?

２進コード binary code: 変換することで本来の形式になる．

命令実行サイクル(CISCマシン)

命令フェッチ(Instruction Fetch, IF) メモリより命令取り出し

命令デコード(Instrunction Decode ID) 命令解析

オペランドフェッチ(Operand Fetch, OF) 命令に使用するデータ取得

実行(Execution, EX) 実行

結果格納(Writeback, WB) 実行結果を格納.

次アドレス決定 通常は次アドレス．分岐命令では，PCに新しいアドレスを格納.

命令実行サイクル(RISCマシン)

各ステージをパイプライン方式で実行．(5章で詳しく説明)

命令フェッチ(Instruction Fetch, IF) メモリより命令取り出し

命令デコード，レジスタフェッチ(Instrunction Decode ID) 命令解析とレジスタから値を読み出す

実行(Execution, EX) 実行して，レジスタに値を格納．分岐の場合はPC書き換え

メモリアクセス(Memory access, MEM)

書込 (WriteBack, WB) メモリからのデータをレジスタに書き込む

基本命令セット

データ命令 算術演算，論理演算，ビット列操作

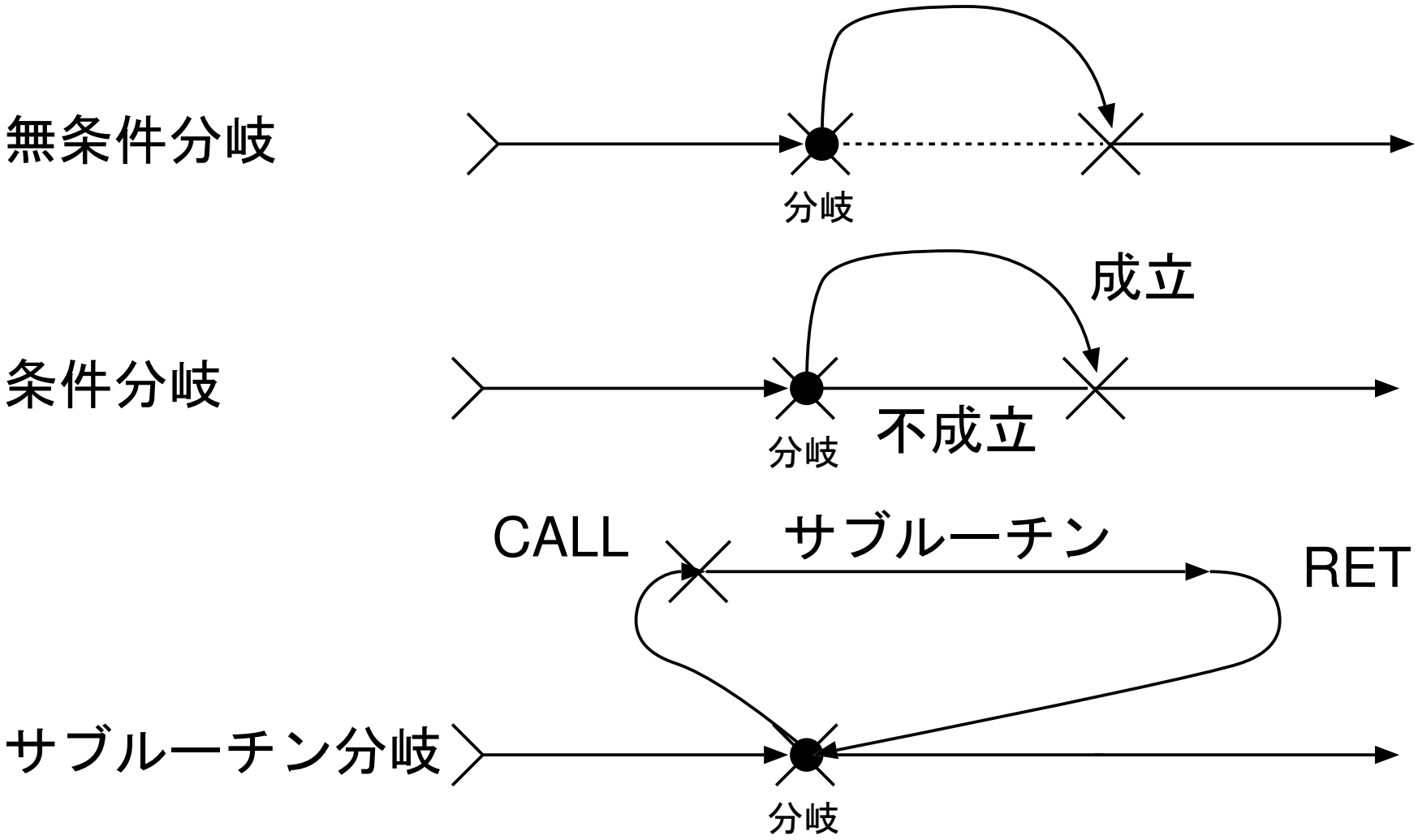
シフト 算術シフト，論理シフト，循環シフト

転送 ロード(メモリ to レジスタ)，ストア(レジスタ to メモリ)

制御命令 無条件分岐，条件分岐(if文)，サブルーチン分岐

特殊命令 IO制御，割り込み，NOP(No Operation)

条件分岐



教科書 p63 図2.33より

条件分岐

分岐する条件として，様々なものが指定できる．

算術演算の結果 if(a-b>0)

論理演算の結果 if(c)

計算機の状態

演算の結果や，計算機の状態は，各種のフラグレジスタに格納され，このレジスタの状態によって，分岐するしないを決める．

プロセッサをどうやって動かす?

プロセッサを動かしているものは?

オペレーティングシステム: OS 人間とコンピュータ間のインタフェース. OS 自身もプログラムのひとつ.

タスク (プロセス) 計算機の中で動くプログラムの単位. OS により実行が制御される.

どうやってプログラムを作る?

記述方法

アセンブラ プロセッサのプログラムコード(機械語)を人間のわかりやすいように記述. ISAに依存する.

高級言語 プロセッサに依存せず, かつ人間が記述しやすい言語を定義して, 機械語に変換する. ハードウエアに直接アクセスする命令を除いて, ISAにほとんど依存しない.

実行方法

コンパイル あらかじめ, 機械語に変換しておく. C言語. コンパイラ: 変換のためのプログラム.

インタプリト その都度プログラムを読み込んで実行する. BASIC. インタプリタ 変換して実行するプログラム

CISCとRISC

CISC時代

- CPUの初期段階の構造は，すべてCISC.
- CISCは，少ないメモリ，レジスタ数で実行できる.
- 多機能な命令が多数存在し，人間が機械語でプログラムしやすい.
- 高級言語の登場により，複雑な命令が使われにくくなってきた.
- ハードウエアの複雑化により，速度向上が難しくなってきた.

RISCの登場

- 速度低下の原因であったメモリを演算処理から切り放す.
- 命令実行ステージをパイプライン化して，高速化
- レジスタを多数配置.
- 同じことを実行する場合の命令列は長くなる.
- 実行プログラム生成は，コンパイラが行なう.

プロセッサの性能

$$TPI = TPC \times CPI$$

- RISCでは，パイプライン化により，TPC(クロック周期)を小さくできる(クロック周波数が高くなる．CPI(命令毎のクロック数)も，パイプライン化により高くなる．ただし，1命令で可能な処理はCISCより小さい.
- TPC×CPIの点でRISCが有利.