初版：2012年8月24日
改訂：2018年7月14日
監修：津田伸生

金沢工業大学

情報工学科

2018 年度版

コンピュータシステム基礎教科書

（後編）

目次

- NT-Processor V1 エミュレータの操作については操作編を参照.
- 毎週，受講前に必要な部分を印刷して持参することを推奨する
- 毎週の授業では各自のノート等に要点のメモをとること
- 毎週の授業の予習・復習を欠かさないこと．特に「6.6 プログラミングの例題と問題」ではエミュレータでプログラムを繰り返し実行し，例題と問題の要点は何かを見直すこと

第６章　16 ビットプロセッサ NT-Processor V1 によるプログラミング

この章では，基本版 V1 のエミュレータ（ハードウェアプロセッサの機能を模擬的に実行するソフトウェア，MS-Excel で作成）を使用して，プロセッサの動作原理と基礎的な機械命令プログラミングを学ぶ.

6.1 プロセッサの概要

NT-Processor V1 はグラフィックス型論理設計システム Visual_Elite で作成した CPU とメインメモリからなる組込みシステム用 16 ビットプロセッサ（正確にはハードウェア記述言語 HDL で定義したプロセッサ用論理ファイル）である．組込みシステムとは，情報家電や産業機器に用いる専用のボード型コンピュータのことで，一般に，電子回路ボードに搭載してある書き換え可能論理 LSI（別名 FPGA：Field Programmable Gate Array）に，上記の論理ファイルを書き込んでハードウェアのプロセッサとして動作させる．（27 ページの図８参照）

NT-Processor V1 は，使用できる機械命令を 25 種類に限定し，メインメモリ容量を 256 ワード（512 バイト）とした教育用バージョン（基本版）で，その上位バージョンとして機械命令をさらに 21 種類追加し，メインメモリ容量を V1 の 256 ワードを１ページとして最大 256 ページまで拡張可能にした実用版プロセッサ V1plus がある．V1plus ではＣコンパイラで作成したプログラムを実行できる．（V1plus はこの授業では使用しない.）

6.2 プロセッサの構成

図１に NT-Processor V1 の機能モジュール構成，図２に NT-Processor V1 の概略機能仕様を示す．プロセッサは次の 8 個の機能モジュール（回路ブロック）からなっている.

- **ALU（算術論理演算ユニット）**：16 ビット（2 バイト）幅．2 進加減算・比較機能，論理演算機能，ビット列シフト機能を備える．V1plus ではさらに 10 進加算・10 進補数変換機能を備える.
- **MM（メインメモリ）**：V1 は 2 バイト×256 ワード（1 ページ）で ROM 領域と RAM 領域からなり，RAM 領域は 16～64 ワードの範囲で可変．V1plus は V1 の 2 バイト×256 ワードを 1 ページとして最大 256 ページ（2 バイト×64k ワード）まで拡張可能.
- **IR（命令レジスタ，2 バイト）**：実行中の機械命令のワードを 1 命令サイクルの期間保持する.
- **IA（命令アドレスレジスタ，1 バイト）**：プログラムカウンタともいう．次に MM から読み出す命令ワードのアドレス（番地）を１命令サイクルの期間保持する．アドレスの歩進（+1）機能をもつ．シーケンサから与えられるデータワード（オペランド）のアドレスは通過させる.
- **CC（コンディションコードレジスタ，2 ビット）**：フラグレジスタともいう．ALU での加算・減算・比較結果の正・負・ゼロ・オーバーフローを 2 ビットの数値で保持し，分岐命令の分岐条件で使えるようにする．下位ビットを C0，上位ビットを C1 で表記する.
- **MDR（メモリデータレジスタ，2 バイト）**：MM からの読み出しデータを一時保持する.
- **GR（汎用レジスタ）**：2 バイト×16 ワードで，ALU での演算結果などを保持する.
- **SEQ（シーケンサ）**：クロック発生回路，命令デーコーダ，オペランド切り出し回路，タイミング同期回路，分岐判定回路からなり，プロセッサ内部の制御信号を生成する.

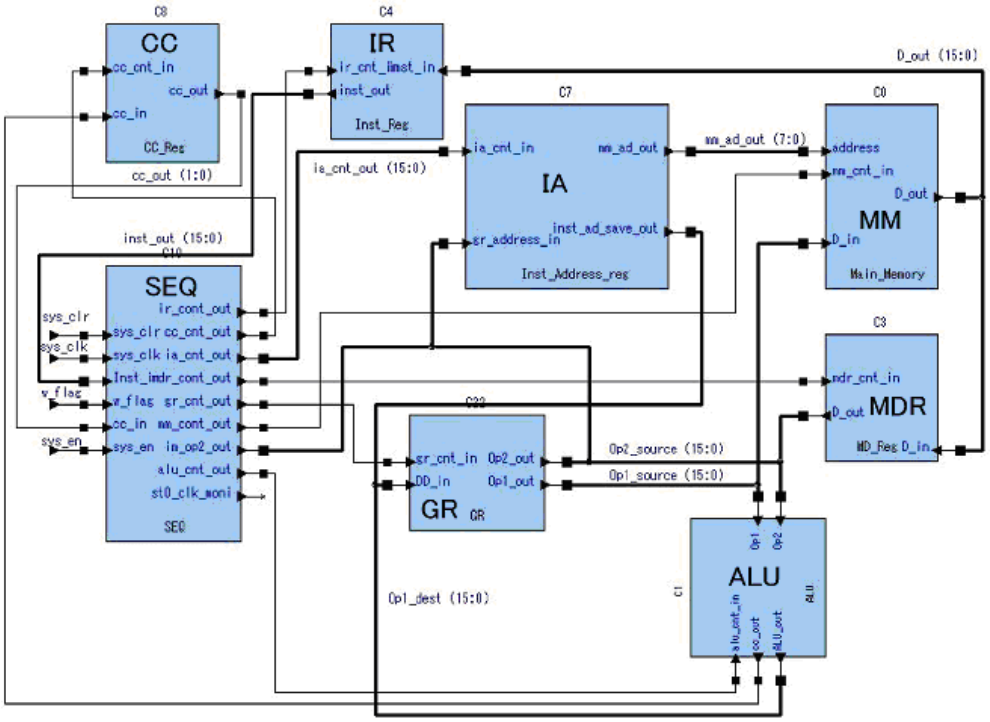


図１　NT-Processor V1 の機能モジュール構成（V1plus もほぼ同一構成）

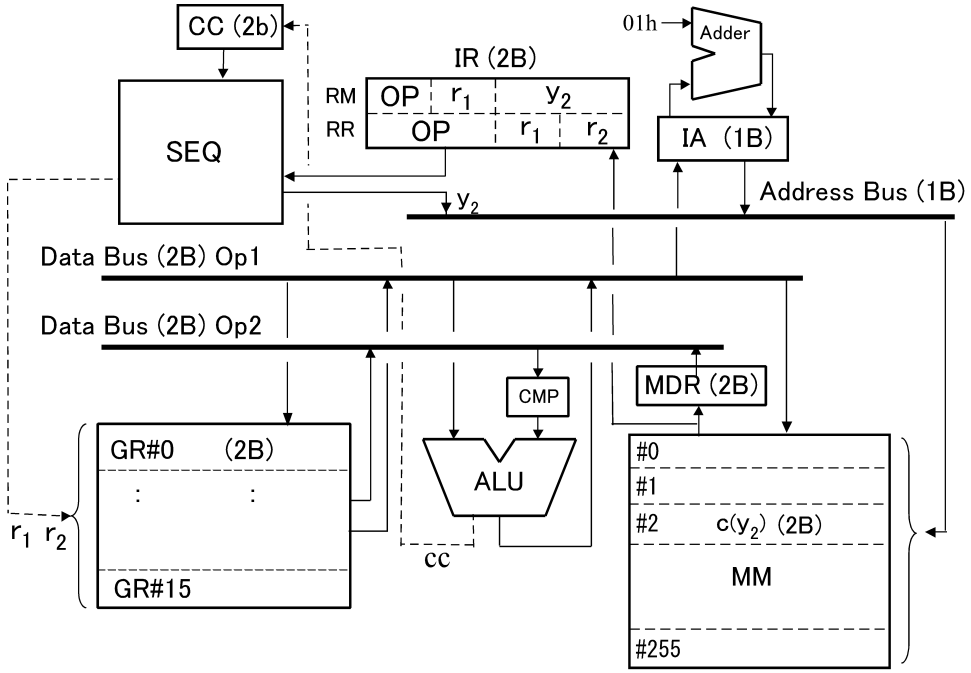


図２　NT-Processor V1 の機能仕様（V1plus の IA は 2B，MM は上記の複数ページ仕様）

図２に示すように，プロセッサ内部の ALU のソースオペランド用データバスは，Op1_bus と Op2_bus の 2 系統からなり，Op1_bus は ALU の演算結果を GR へ格納するためにも用いられる（2・1/2 バス構成）．MM からの読み出しデータは MDR を経由して Op2_bus へ送られ，逆に MM への書き込みデータは GR から Op1_bus を経由して MM へ送られる．これらのバスの使い方は，6.3 で述べる機械命令の第 1 オペランド，第 2 オペランドにそのまま対応している．

このような NT-Processor V1 プロセッサのアーキテクチャ（構成法）は，現在のプロセッサのルーツとなった IBM System/370 32 ビットメインフレームを参考に, 独自のコンセプトを加えて 16 ビットで再構成したものである．以下にその特徴をまとめる．

- すべての機械命令のワードは 2 バイト（16 ビット）固定長
- ２オペランド形式でバス構造と内部制御が単純
- 機械命令の命令ワードでオペランドが明示されている
- 機械命令の命令コード（OP コード）の 16 進数から処理内容（オペレーション）を連想でき，アセンブラ表記なしでも直接機械命令でプログラムを記述できる
- 汎用レジスタ GR の 16 ワードを制約なしで使えるため，プログラミングが容易
- 演算結果のコンディションコード CC と分岐条件を示す条件マスク(m_code)の関係が単純

注) 通常のプロセッサは 1～3 バイト長の機械命令を使用している．このため機械命令でプログラムを記述するのは困難で，アセンブラ表記で行う必要がある．8 ビットプロセッサ(マイコン)では，演算結果はアキュムレータと呼ぶ特定のレジスタに入るようになっている．

6.3 命令ワードの構成

このプロセッサで使用するすべての機械命令のワードは前述のように 2 バイト長(16 ビット長）で, 各命令ワードは命令コード（Op_code）, 第 1 オペランド（Op_1), 第 2 オペランド（Op_2) からなっている. 機械命令の記述形式には RR 形式と RM 形式の 2 種類がある（RR: register and register, RM: register and memory). 下に 1A 命令（レジスタ加算命令）と A 命令（加算命令）の例を示す.

RR 形式命令は，8 ビット の命令コード，第 1 オペランドの GR（汎用レジスタ）のワード番号を 4 ビットで指定する **r1**，第 2 オペランドの GR のワード番号を 4 ビットで指定する **r2** からなっている（**r** は **r**egister を指す）．使用可能な GR のワード数は 16 で，ワード番号 0～F で指定する．

RM 形式命令は，4 ビットの命令コード，第 1 オペランドの GR のワード番号を 4 ビットで指定する **r1**，第 2 オペランドのメインメモリ MM のアドレス（番地）を 8 ビットで指定する **y2** からなっている（**y** は memor**y** を指す）．（注：9 命令のみ y2 に代えて 8 ビット即値"x"を記述できる．詳細は後述）

プロセッサで実行するプログラムは，メインメモリの ROM 領域に機械命令ワードを書き込んで定義する．

■ RR 形式

例）1A 命令（レジスタ加算命令）: 1A　r1　r2　（アセンブラ表記 : AR　r1, r2）

命令コード＋第 1 オペランド＋第 2 オペランド
(8-bit)　(4-bitGR 番号)　(4-bitGR 番号)

■ RM 形式

例）A 命令（加算命令）:　A　r1　y2　（アセンブラ表記 : A　r1, y2）

命令コード＋第 1 オペランド＋第 2 オペランド
(4-bit)　(4-bitGR 番号)　(8-bitMM アドレス)

注）アセンブラ表記の命令コード AR や A をニーモニック（Mnemonic）という．A 命令の場合は機械命令の命令コードもニーモニックも A.

プロセッサのハードウェアでは，RR 形式命令の命令サイクル（逐次制御手順）は 4 ステージ（4 クロック），RM 形式は 6 ステージで演算を実行する．なお，メモリアクセスを行わない一部の RM 形式命令（7 命令， 9 命令，詳細は後述）は RR 形式と同じ 4 ステージで動作する．

6.4 命令セット

以下に，V1 用の 25 種類の機械命令を命令コード順（分岐命令は後述）に解説する．（命令セットの要約は，別資料「Instruction table (V1_ad16)」またはエミュレータの「Instruction table」シートを参照）．**注）命令機能の英文中の青文字は命令コードを覚える際に利用できる．**

演算結果を第 1 オペランドの GR ワード r1 に格納する命令では，演算前の r1 の内容 c(r1)は保存されない点に注意．（注：c(　) は content（内容）を表す）

6.4.1 RR 形式命令（特殊命令）

*以下の命令において **don't care** とは数値指定に意味がないことを指す．

・00 命令　（Nop: No operation，空命令）　00 r1 r2　(r1, r2 は don't care)
　　IA のアドレス（番地）が＋1 歩進するのみ
　　（通常は 0000 と記述）

・01 命令　（HLT: End Stop，終了停止命令）　01 r1 r2　(r1, r2 は don't care)
　　IA のアドレスを歩進しない
　　（通常は 0100 と記述）

・11 命令　（Wait: Wait until w-flag becomes '0'，待機命令）11 r1 r2　(r1, r2 は don't care)
　　w-flag が'1'のとき：　IA のアドレスは歩進しない
　　w-flag が'0'のとき：　IA のアドレスが＋1 歩進する **（通常は 1100 と記述）**

6.4.2 RR 形式命令（データロード・算術演算用）

・16 命令　（RDM: Load Random Value，乱数ロード命令）　16 r1 r2　(r2 は don't care)
　　r1 ← 8-bit random value
　　（通常は 16 r1 r1 と記述）

・18 命令　（LR: Load Register，レジスタロード命令）　18 r1 r2
　　r1 ← c(r2)

・1A 命令　（AR: Add Register，レジスタ加算命令）　1A r1 r2
　　r1 ← c(r1) + c(r2)

・1B 命令　（SR: Subtract Register，レジスタ減算命令）　1B r1 r2
　　r1 ← c(r1)－c(r2)

・1C 命令　（CR: Compare Register，レジスタ比較命令）　1C r1 r2
　　｜c(r1)｜－｜c(r2)｜　（2-byte 絶対値の数値的大小関係）
　　比較結果が ＝ なら CC = 0，＜ なら CC = 1，＞ なら CC = 2 にセットされる
　　c(r1)は保存される

*1A, 1B 命令では演算に加えてコンディションコード CC が次のようにセットされる．
　演算結果がゼロなら CC = 0，負なら CC = 1，正なら CC = 2，オーバーフローなら CC = 3.
*16 命令の乱数は，プロセッサハードウェアでは“RUN”ボタンまたは“RW”ボタンを押している時間に応じてカウンタ回路で生成される．エミュレータでは Excel の乱数生成機能を使用．

6.4.3　RR 形式命令（論理演算用）

・20 命令　（OR: Logical OR，論理和命令）　　20 r1 r2
　　r1 ← c(r1) or c(r2)

・21 命令　（INV: Invert，論理否定命令）　　21 r1 r2
　　r1 ← inv. c(r2)

・2A 命令　（AND: Logical AND，論理積命令）　　2A r1 r2
　　r1 ← c(r1) and c(r2)

・2E 命令　（XOR: Exclusive OR，排他的論理和命令）　　2E r1 r2
　　r1 ← c(r1) xor c(r2)

6.4.4　RR 形式命令（ビット列シフト用）

　*以下のシフト命令では　r1 ← shifted c(r2)　　であるが，c(r2)=(b_{15}～b_0)$_2$としてシフト操作の内容を示す．

・22 命令（SU: One-bit Shift Up，1 ビット上位（左）シフト命令）　　22 r1 r2
　　r1 ← (b_{14}～b_0 0)$_2$　　LSB には‘0’が入る
　　C1 C0 ← (0 b_{15})$_2$　　オーバーフローした b_{15}が CC の下位ビット C0 に入る
　　CC の上位ビット C1 は‘0’になる
　　（注：CC は 0 または 1 になる）

・23 命令（SD: One-bit Shift Down，1 ビット下位（右）シフト命令）　　23 r1 r2
　　r1 ← (0 b_{15}～b_1)$_2$　　MSB には‘0’が入る
　　C1 C0 ← (0 b_0)$_2$　　アンダーフローした b_0が CC の下位ビット C0 に入る
　　CC の上位ビット C1 は‘0’になる
　　（注：CC は 0 または 1 になる）

・24 命令（ESU: One-bit End-around Shift Up，1 ビット上位回転シフト命令）　24 r1 r2
　　r1 ← (b_{14}～b_0 b_{15})$_2$　　ビット列が 1 ビット繰り上がり LSB には b_{15}が入る

・25 命令（ESD: One-bit End-around Shift Down, 1 ビット下位回転シフト命令）　25 r1 r2
　　r1 ← (b_0 b_{15}～b_1)$_2$　　ビット列が 1 ビット繰り下がり MSB には b_0が入る

6.4.5　RM 形式命令

・8 命令　（L: Load，ロード命令）　　8 r1 y2　　(y2 は MM のアドレス)
　　r1 ← c(y2)

・9 命令　（LA: Load Address，アドレスロード命令）　　9 r1 "x"　　("x"は 8 ビット即値)

r1 ← "x"

*この命令は本来 r1 で指定する GR のワードの下位バイトにアドレス y2 を設定する命令であるが，このことを一般化して 8 ビット即値（Immediate value）"x"を設定するために使用する．r1 の上位バイトは 00h になる．

（記述は RM 形式であるがメモリアクセスを行わないため実行は 4 ステージ）

・A 命令　（A: **A**dd, 加算命令）　　A r1 y2

r1 ← c(r1) + c(y2)

・B 命令　（S: Su**b**tract, 減算命令）　　B r1 y2

r1 ← c(r1)－c(y2)

・C 命令　（C: **C**ompare, 比較命令）　　C r1 y2

｜c(r1)｜－｜c(y2)｜　　（2-byte 絶対値の数値的大小関係）

比較結果が = なら CC = 0, < なら CC = 1, > なら CC = 2 にセットされる

c(r1)は保存される

・D 命令　（ST: Store, ストア（格納）命令）　　D r1 y2

y2 ← c(r1)

*A 命令と B 命令では演算に加えて CC が次のようにセットされる．

演算結果がゼロなら CC = 0，負なら CC = 1，正なら CC = 2，オーバーフローなら CC = 3.

*RM 形式命令では即値"x"またはアドレス y2 に替えてラベルを記述できる．（詳細は後述）

6.4.6　分岐命令

・7 命令　（BC: Branch on Condition, 条件分岐命令）　　7 m y2

分岐ならば　　IA ← y2　when branch (jump)

分岐しないならば　　00 命令（空命令）と同じ

（注：記述は RM 形式であるがメモリアクセスを行わないため実行は 4 ステージ）

分岐条件は m の値（条件マスク m-code という）により次のように定義される

m=0　→ 分岐なし（00 命令（空命令）と同じ）

m=1　→ CC=3　（オーバーフロー）で分岐

m=2　→ CC=2 (>, +) で分岐

m=4　→ CC=1 (<,　－) で分岐

m=6　→ CC=2 or 1 (≠: not=) で分岐

m=8　→ CC=0 (=, 0) で分岐

m=A　→ CC=2 or 0 (> or =, + or 0) で分岐

m=C　→ CC=0 or 1 (= or <, 0 or　－) で分岐

m=F　→ 無条件分岐（強制分岐，F は Forced を意味する）

注) 機械命令では条件を満足すると分岐するが，C プログラムの if 文などでは満足しない場合に分岐する．

・05 命令（BALR: Branch and Link Register, レジスタ連携分岐命令）　　05 r1 r2

r2≠0 のとき:　r1 ← c(IA),　IA ← c(r2)

r2＝0 のとき:　r1 ← c(IA)　　分岐先アドレス等のセットに用いる

・07 命令　（BCR: Branch on Condition to Register, レジスタ条件分岐命令）　07 m r2

r2≠0 のとき: 分岐ならば　　IA ← c(r2)

分岐しないならば　00 命令（空命令）と同じ

r2＝0 のとき:　00 命令（空命令）と同じ

*条件分岐を分岐なしに切り換える際に用いる

*分岐条件は 7 命令と同様に条件マスク m で指定できるが，07 命令では通常 m=F としてサブルーチンからの無条件(強制)復帰に用いる

* 05 命令と 07 命令は次のようにしてサブルーチンコールに用いられる.

メインプログラム

↓

98 "x"　(サブルーチン先頭番地 y2 を"x"として GR#8 にセット)

05 98　(GR#8 のサブルーチン先頭番地"x"(y2)を IA にロード,

↓　メインルーチンへの復帰番地を GR#9 にセット)

:

サブルーチン先頭

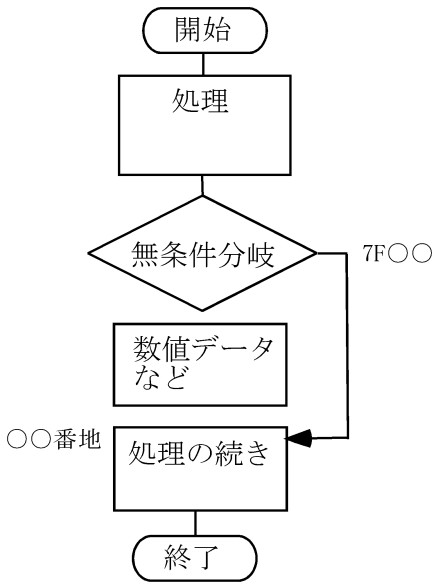
↓

07 F9　(サブルーチン終了：GR#9 の復帰番地へ無条件分岐(m=F))

6.4.7　フローチャート

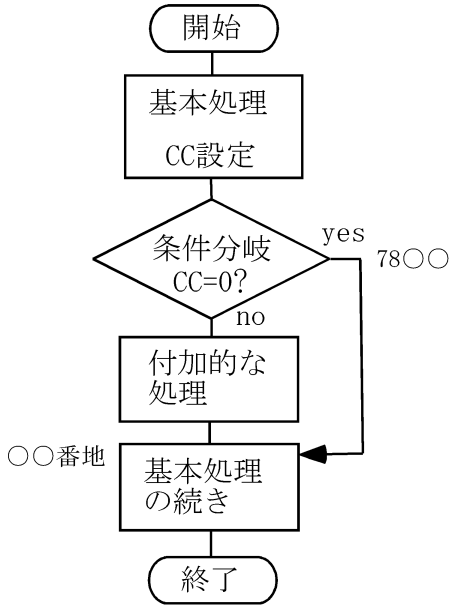
以下に分岐命令の使用例をプローチャート（流れ図）で示す.（注：ここでは分岐命令のみを判断ボックス(菱形)で示す.）

無条件(強制)分岐命令の使い方



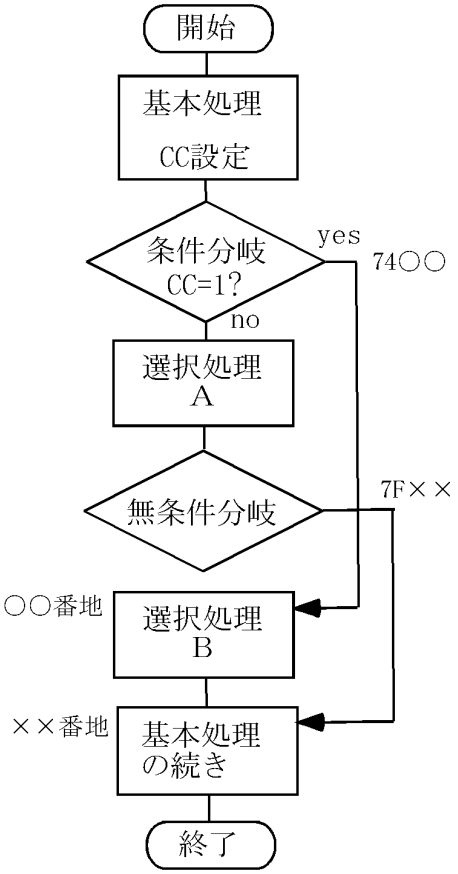
条件分岐命令の使い方（単純分岐型）

ある演算結果が非ゼロのとき付加的な処理を実行する例



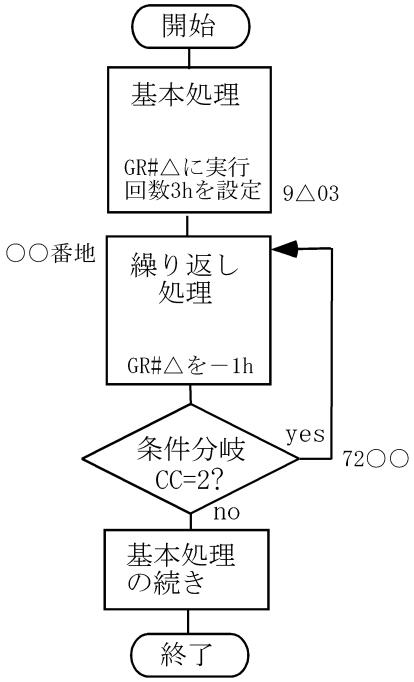
条件分岐命令の使い方（選択実行型）

ある演算結果がゼロまたは正のとき選択処理Ａを実行し、
負のときは選択処理Ｂを実行する例



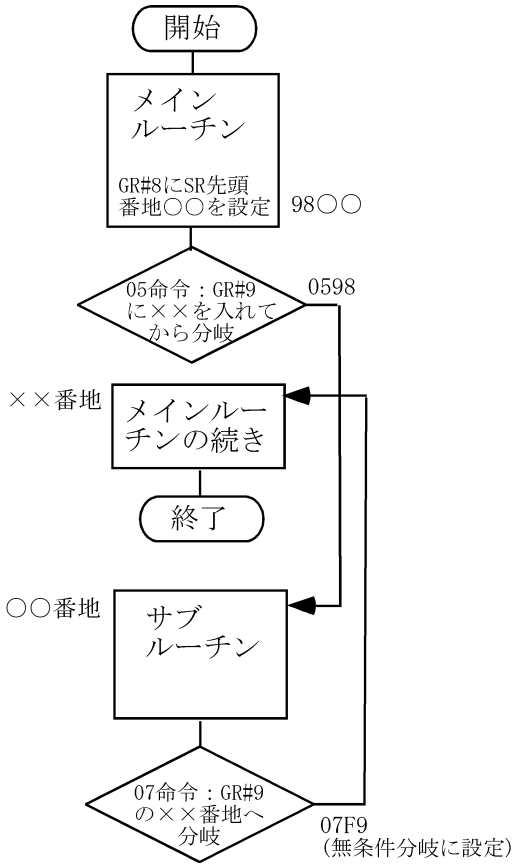
条件分岐命令の使い方（繰り返し処理：ループ処理）

繰り返し処理部を３回実行する例



レジスタ連携分岐命令(05命令)・レジスタ条件分岐命令(07命令)によるサブルーチンコール

GR#8にサブルーチン先頭番地，GR#9に復帰番地を入れる例



6.4.8 ラベル

V1 エミュレータでは，第２オペランドにメインメモリ MM のアドレス（番地）y2 を記述する RM 形式の命令において，y2 に代わって相対的にプログラム行位置を示す記号であるラベルを使用できる．また 9 命令「アドレスロード命令」では即値"x"に代わってラベルを書くとそのラベルに対応するアドレス y2 が r1 で指定する GR のワードに書き込まれる．r1 の上位バイトは 00h になる．

ラベルを用いるとプログラムを再配置可能（リロケータブル）にできるとともに，プログラムの解析を容易にする効果が得られる．

ラベルは英数字２文字からなり，第１文字は a～f，l(エル)，o(オー)，r，y 以外のアルファベット小文字，第２文字は 0～9，A～F の 16 進数字，または第１文字と同様なアルファベットとする．（ラベルの使用方法の詳細はエミュレータの「Instruction Table」シートを参照）

6.5 エミュレータの操作とプログラムの記述

6.5.0 予備演習

NTP-V1-emulator を用いて以下の予備演習を行う．

◎ステップ１：エミュレータ操作とプログラムの流れを理解する

(1)「Instruction Table」シートで Test Program 0 を実行する．

(2)「10-16-2 conv」シートで 10 進数-16 進数-2 進数変換およびその逆を体験する．また，10 進数入力による 16 進数四則演算を体験する．

(3)「Mul_prog（乗算)」,「Div_prog（除算)」の各シートで計算手順（アルゴリズム）を考察する．（乗算，除算アルゴリズムは補足編の 6，7 ページに解説してある．）

(4)「Dec-Hex (10 進数-16 進数変換)」,「Calc (電卓処理)」,「PrimeNum Prog (素数探索)」の各シートでプログラムを実行する．

◎ステップ２：機械命令の構造と種類を理解する

(5)「Inst_Exercise」シートで RL 形式の Test Program 0 の命令ワード入力を練習する．

(6)「Mul_Exercise（乗算演習)」,「Div_ Exercise（除算演習)」の各シートで命令ワードの穴埋めを行いプログラムを完成させる．完成したプログラムの実行ステップ数を「Mul_prog（乗算)」,「Div_prog（除算)」のものと比較する

6.5.1 例題･問題の表記について

・汎用レジスタ GR の操作： 正式には GR#0 ← c(GR#A) + c(GR#B) とその内容を表す c( ) を用いて書くが，ここでは c( )を省略して

GR#0 ← GR#A + GR#B　　と記述する．

・メインメモリ MM の操作： 正式には F0h ← c(1Ch) + c(1Dh) と番地（アドレス）のみ，またはその番地のワード内容を表す c( )を付けて書くが，ここでは MM のワードを示す[ ]を付けて

[F0h] ← [1Ch] + [1Dh]　　と記述する．

ラベルを用いる場合は　[x0] ← [v1] + [v2]　　と記述する．

6.5.2 メインメモリの構成

NT-Processor V1 は FPGA を用いる組込みプロセッサであるため, メインメモリ MM はプログラムを書き込む ROM 領域（読み出しのみ）と演算結果を書き込む RAM 領域（読み出し／書き込み可）に分かれている.

以降のプログラムの説明では，次の構成とする（エミュレータと同じ）.

ROM 領域（読み出しのみ）: [00h]～[EFh]

RAM 領域（読み出し／書き込み可）: [F0h]～[FFh]

6.5.3 プログラムの記述

例１） GR#0 ← GR#A + GR#B （GR#A , GR#B の内容は保存）

プログラムの先頭は 10h 番地とする.

番地	機械命令	アセンブラ表記	コメント
10	180A	LR 0, A	GR#0 ← GR#A
11	1A0B	AR 0, B	GR#0 ← GR#0+GR#B
12	0100	HLT	終了停止

例２） [F0h] ← [1Ch] + [1Dh]

プログラムの先頭は 10h 番地とする．GR#A を演算に用いる.

番地	機械命令	アセンブラ表記	コメント
10	8A1C	L A, 1C	GR#A ← [1Ch]
11	AA1D	A A, 1D	GR#A ← GR#A+[1Dh]
12	DAF0	ST A, F0	[F0h] ← GR#A
13	0100	HLT	終了停止
:	:	:	
1C	1111 v	1111 v	数値データ
1D	2222 v	2222 v	数値データ

例３） [x0] ← [v1] + [v2]

プログラムの先頭は 10h 番地とする．GR#A を演算に用いる.

番地	ラベル	機械命令	ラベル	アセンブラ表記	コメント
10		8A v1		L A, v1	GR#A ← [v1]
11		AA v2		A A, v2	GR#A ← GR#A+[v2]
12		DA x0		ST A, x0	[x0] ← GR#A
13		0100		HLT	終了停止
:		:		:	
1C	v1	1111 v	v1	1111 v	数値データ
1D	v2	2222 v	v2	2222 v	数値データ

◎ プログラムを記述する際のルール

- プログラムでは，16 進数 2 桁（8 ビット）で指定される番地（アドレス）の末尾の"h"は番地の列，アセンブラ表記の番地のオペランドとも省略する．ただし，コメントでは"h"をつける.

- 番地の列は省略可（必要な場合のみ書く）.
- 機械命令の命令ワードは OP コード，第 1 オペランド，第 2 オペランドからなる 16 進 4 桁の数字を並べて書く．第 2 オペランドがラベルの場合もその 2 文字を同様に並べて書く.
- アセンブラ表記の命令ワードは，ニーモニックの右にスペースをあけ，第 1 オペランドと第 2 オペランドをカンマ","で区切って書く．オペランドの"GR#"や"[ ]"は省略する．コメントでは"GR#"や"[ ]"をつける．（注：エミュレータでは，機械命令を入力するとアセンブラ表記を自動表示する機能があるが，オペランドには"GR#"，"MM-"，"lbl-"をつけている）
- アセンブラ表記の命令ワードでは，01 命令（HLT）のように機械命令でのオペランド指定に意味がない場合はニーモニックのみを書く.
- ラベルは機械命令，アセンブラ表記とも命令ワードの左に書く.
- 数値データを示す"v"サインは機械命令，アセンブラ表記ともデータワードの右に書く.

6.5.4 プログラム実行追跡表

プログラムを実行した際のレジスタやメモリワードの状態（状態）の変化を記述した表をプログラム**実行追跡表**という（単に追跡表ともいう）.

例１） GR#0 ← GR#A + GR#B　（GR#A ， GR#B の内容は保存）

プログラムの先頭は 10h 番地とする.

	IA	IR	CC	GR#0	GR#A	GR#B	アセンブラ表記	コメント
初期値	10				1111	2222		
	11	180A		1111	1111	2222	LR　0, A	GR#0 ← GR#A
	12	1A0B	2	3333	1111	2222	AR　0, B	GR#0 ← GR#0+GR#B
	12	0100	2	3333	1111	2222	HLT	終了停止

例２） [F0h] ← [1Ch] + [1Dh]

プログラムの先頭は 10h 番地とする．GR#A を演算に用いる.

	IA	IR	CC	GR#A	[1Ch]	[1Dh]	[F0h]	アセンブラ表記	コメント
初期値	10				1111	2222			
	11	8A1C		1111	1111	2222		L　A, 1C	GR#A ← [1Ch]
	12	AA1D	2	3333	1111	2222		A　A, 1D	GR#A ← GR#A+[1Dh]
	13	DAF0	2	3333	1111	2222	3333	ST A, F0	[F0h] ← GR#A
	13	0100	2	3333	1111	2222	3333	HLT	終了停止

例３） [x0]　← [v1] + [v2]

プログラムの先頭は 10h 番地とする．GR#A を演算に用いる.

	IA	IR	CC	GR#A	[v1]	[v2]	[x0]	アセンブラ表記	コメント
初期値	10				1111	2222			
	11	8Av1		1111	1111	2222		L　A, v1	GR#A ← [v1]
	12	AAv2	2	3333	1111	2222		A　A, v2	GR#A ← GR#A+[v2]
	13	DAx0	2	3333	1111	2222	3333	ST A, x0	[x0] ← GR#A
	13	0100	2	3333	1111	2222	3333	HLT	終了停止

◎実行追跡表を記述する際のルール

- 表の第１行に，左から IA，IR，CC，使用する GR のワード番号，使用する MM のワードの番地またはラベル，アセンブラ表記（IR に読み込まれた機械命令ワードに対応する），コメントの欄を設ける．
- 表の第２行に，初期値を書く欄を設ける．初期値がない場合は空欄にしておく．
- 表の第３行以下に，IR に読み込まれた機械命令ワードの実行によって変化した各レジスタや MM ワードの状態を，上から下へ命令の実行順に書く．状態が変化しない場合は空白の場合を含めて上の状態をそのまま書く．

注）第３行以下の各行は，IR に読み込んだ命令を実行した後の状態を示している．IA の内容は次に読み出す命令ワードの番地になる．分岐命令では，分岐ありの場合に分岐先番地に書き換えられる．01 命令「終了停止命令：HLT」では IA の番地は歩進（+1）しない点に注意．

6.6 プログラミングの例題と問題

演習手順：エミュレータの以下のシートを使用してプログラミングの方法を理解する．

(1) **命令の基本的な使い方（6.6.0）：Inst_Functions** シートの各プログラムを実行する．シート内の理解度確認問題を口頭で解答しチェックする．不明な場合は，16 ページの解答例を読んで理解する．（注：Inst_Functions (2) は命令コードを空白にした再演習用シート）

(2) **命令のやや高度な使い方（6.6.0）：Inst_Advance** シートの各プログラムを実行する．シート内の理解度確認問題を口頭で解答しチェックする．不明な場合は，19 ページの解答例を読んで理解する．（注：Inst_Advance (2) は命令コードを空白にした再演習用シート）

(3) **命令の基本的な使い方の応用（6.6.1～6.6.2）：Prog_Exercise1 シート**の例題プログラムを実行し，問題プログラムの穴埋めを行う．シート内の理解度確認問題を口頭で解答しチェックする．不明な場合は，21 ページの解答例を読んで理解する．**（注：この演習(3) は演習(2) 実施前でも可能）**

(4) **命令のやや高度な使い方の応用（6.6.3～6.6.6）：Prog_Exercise2～4 シート**の例題プログラムを実行し，問題プログラムの穴埋めを行う．各シート内の理解度確認問題を口頭で解答しチェックする．不明な場合は，23 ページ以降の解答例を読んで理解する．

（注１：以上を繰り返し実行して理解を深めること）

（注２：Inst_Functions(2)，Inst_Advance(2)，Prog_Exercise1(2)，Prog_Exercise2(2)の各シートは命令コードのほぼすべてを空白にした再演習用．）

6.6.0 命令の使い方

◎ **Inst_Functions シート（NT-ProcessorV1 用基本命令の使用例）**

以下，使用例の題目のみを示す．具体的なプログラムはエミュレータのシートを参照．

[IF1] データ移動命令（ロード命令：８，レジスタロード命令：18，ストア(格納)命令：D）

[IF2] ロード命令：８，ストア(格納)命令：D，ラベルによる番地指定（RL 形式）

[IF3] 即値代入命令（アドレスロード命令：９）

◎その他の数値ロード方法

[IF4] 加算，減算，比較命令（RM形式：A，B，C命令），CC に注意

[IF5] レジスタ加算，レジスタ減算，レジスタ比較命令（ＲＲ形式：1A，1B，1C 命令），CC に注意

[IF6] 論理演算命令（ＲＲ形式: OR 命令:20，AND 命令:2A，XOR 命令:2E，INV 命令:21 命令）

[IF7] ビット列シフト（ＲＲ形式: 上位回転: 24，下位回転: 25，上位: 22，下位: 23 命令）

[IF8] 分岐命令（７命令：無条件（強制）分岐）：数値データワードを無条件分岐で迂回する例

[IF9-1] 分岐命令（７命令：条件分岐）GR#0 の値が奇数なら GR#0 を＋１する例（分岐なしの場合）

[IF9-2] GR#0 の値が奇数なら GR#0 を＋１する例（上記と同じ処理で分岐ありの場合）

[IF10-1] GR#0 の値が奇数なら GR#0 を＋１，偶数なら－１する例（奇数の場合）

[IF10-2] GR#0 の値が奇数なら GR#0 を＋１，偶数なら－１する例（上記と同じ処理で偶数の場合）

[IF11] 繰り返し処理（ループ処理）（条件分岐命令を用いた実行回数制御）

GR#0 に GR#1 の値を３回加算する例：GR#C をカウンタとして用いる

[IF12] サブルーチンコール（レジスタ連携分岐命令:05，レジスタ条件分岐（復帰）命令:07）

サブルーチン内で GR#0 に数値をロードしその値を２倍する例

[IF13] GR#8 と GR#9 を用いてサブルーチン内からさらに他のサブルーチンをコールする例

第１階層サブルーチン内で GR#0 へ数値をロードし，第２階層サブルーチン内でその値を２倍する例

◎ Inst_Functions シート理解度確認問題

次の問いに答えよ．

(1) プロセッサ（CPU）で機械命令プログラムを実行するにあたり，IR（命令レジスタ）と IA（命令アドレスレジスタ）は何をするためにあるのか．

IA が指定する MM の番地から次に実行する命令ワードを読み出して IR に書き込む．命令ワードの実行後，IA の内容は次に読み出す命令の番地になる．

(2) プログラム実行時に表示される黄色，青色，灰色の各マーカーは何を示しているか．

黄色：現在実行中の命令ワード，青色：第１オペランド，灰色：第２オペランド．

(3) ラベルの役割について説明せよ．

プログラムで MM の番地に替えてこれを書くことによりプログラムを再配置可能（RL 形式）にする．また，プログラムの解析を容易にする．

(4) GR のワードに１バイトの数値を設定する方法を２つ挙げよ．また，２バイトの数値を GR のワードに設定するにはどのようにするのか．

9 命令「アドレスロード命令」の第 2 オペランドに 1 バイト即値を書いて設定するか，上位バイトを 00h にしたデータワードをプログラムに書いておき 8 命令「ロード命令」でロードする．2 バイトの数値は 8 命令で 2 バイトを有効にしてロードする．（[IF3] 参照）

(5) RM 形式命令は第２オペランドに即値を書くものを含めて何種類あるか．また，それらをすべて挙げよ．

7 種類：7 命令「条件分岐命令」，8 命令「ロード命令」，9 命令「アドレスロード命令」，A 命令「加算命令」, B 命令「減算命令」, C 命令「比較命令」, D 命令「ストア命令」

(6) 加算，減算，比較には，それぞれ RR 形式命令と RM 形式命令が用意されているが，どのように使い分けるのか．

RR 形式命令は GR の 2 つまたは 1 つのデータワードの演算，RM 形式命令は GR のデータワードと MM のデータワードの演算に用いる．（[IF4], [IF5] 参照）

(7) 論理演算用の命令は何種類あるか．また，それらの論理機能を述べよ．

4 種類：20 命令「論理和(OR)命令」第 1 オペランドと第 2 オペランドでビット単位で OR をとり第 1 オペランドへ格納する，21 命令「論理否定(INV)命令」第 2 オペランドのビット値を反転させ第 1 オペランドへ格納する，2A 命令「論理積(AND)命令」第 1 オペランド

と第 2 オペランドでビット単位で AND をとり第 1 オペランドへ格納する，2E 命令「排他的論理和(XOR)命令」第 1 オペランドと第 2 オペランドでビット単位で XOR をとり第 1 オペランドへ格納する．（[IF6] 参照）

(8) すでに数値が設定されている GR ワードをゼロクリアする（all '0'にする）方法を 4 つ挙げよ．

1) 9 命令「アドレスロード命令」の第 2 オペランドに 1 バイト即値 00h を書いて設定する．（[IF3] 参照）
2) プログラムに数値データワード 0000h を定義しておき 8 命令「ロード命令」でロードする．（[IF3] 参照）
3) 1B 命令「レジスタ減算命令」で第 1 オペランドと第 2 オペランドに同じ GR ワードを指定する．（[IF5] 参照）
4) 2E 命令「排他的論理和(XOR)命令」で第 1 オペランドと第 2 オペランドに同じ GR ワードを指定する．（[IF6] 参照）

(9) プログラムの分岐を行うための命令をすべて挙げよ．

7 命令「条件分岐命令」（通常の分岐用），05 命令「レジスタ連携分岐命令」（サブルーチンへの分岐用），07 命令「レジスタ条件分岐命令」（サブルーチンからの復帰用）．

(10) 7 命令は，プログラム実行時にどのようにして分岐先番地を設定するのか．

第 2 オペランド部に定義してある MM の分岐先番地（またはラベル）を IA に設定する．（[IF9-1], [IF9-2], [IF10-1], [IF10-2] 参照）

(11) 条件分岐命令は何に基づいて分岐あり／分岐なしを決めるのか．

先行命令で設定したコンディションコード（CC）の値と条件分岐命令に定義してある条件マスク（m コード）の値から分岐あり／分岐なしを決める．（[IF9-1], [IF9-2], [IF10-1], [IF10-2] 参照）

(12) 条件分岐命令では，分岐先番地は分岐あり／分岐なしの場合にどのようにマーカーで表示されるか．

分岐ありの場合は分岐先番地に茶色，分岐なしの場合は灰色のマーカーが表示される．

(13) 繰り返し処理（ループ処理）において実行回数を決める手順を述べよ．

GR ワードに実行回数の数値を設定しておき，ループの実行ごとにその値を－1 して実行回数がゼロになったかどうかを条件分岐命令で判定する．ゼロの場合はループを抜ける．（[IF11] 参照）

(14) GR のワードに設定された数値データの偶数／奇数を判定する手順を述べよ．

23 命令「1 ビット下位シフト命令」で GR ワードの LSB をコンディションコードレジスタ（CC）に入れ，7 命令「条件分岐命令」でその値が'1 なら奇数，0 なら偶数と判定する．（[IF9-1], [IF9-2], [IF10-1], [IF10-2] 参照）

(15) GR のワードに，あるラベルが示す番地を設定する手順を述べよ．

9 命令「アドレスロード命令」の第 2 オペランドにラベルを書いて実行すると，プログラム内そのラベルが書かれた位置の番地が第 1 オペランドの GR ワードに設定される．（[IF3] 参照）

(16) サブルーチンコールでの分岐と復帰はどの命令でどのようにして実行されるのか．

メインルーチンにレジスタ連携分岐命令「05 命令」を置くとそこでサブルーチンへ分岐し，サブルーチンの末尾に m コードを強制分岐 F にしたレジスタ条件分岐命令「07 命令」置くとメインルーチンの 05 命令の次の番地へ復帰する．05 命令では第 2 オペランドが指定する GR ワードに予めサブルーチンの先頭番地を設定しておく．サブルーチンからの復帰

番地は 05 命令の第 1 オペランドが指定する GR ワードに自動で設定される．07 命令では第 2 オペランドにこの GR ワード番号を指定しておく．（[IF12] 参照）

(17) 2 つの GR ワードのみを用いて，あるサブルーチン内から他のサブルーチンをコールするにはどのような手順が必要か.

分岐先番地用 GR ワード番号と復帰番地用 GR ワード番号各一つを指定する．メインルーチンから第 1 階層サブルーチンへは通常の方法で分岐する．分岐後に復帰番地用 GR ワードの内容を MM へ退避させるとともに，分岐先番地用 GR ワードに第 2 階層サブルーチンの先頭番地を設定して第 2 階層サブルーチンへ分岐する．通常の方法で第 1 階層サブルーチンへ復帰した後に MM に退避してあるメインルーチンへの復帰番地を復帰番地用 GR ワードに書き戻してメインルーチンへ復帰する．（[IF13] 参照）

◎ Inst_Advance シート（NT-ProcessorV1 用基本命令による機能拡張）

[IA1] V1 用基本命令のみによる連続番地からのデータロード・連続番地へのストア(格納)処理

ROM 領域 B0 番地以下の 3 つの数値データを RAM 領域 F0 番地以下の 3 つのワードへ格納する例

GR のワードにロード命令，ストア命令を設定しその番地を歩進(+1)しつつ RAM 領域のワードへコピーしてそこへ分岐する. RAM 領域にコピーした命令の次には復帰用の分岐命令をおく.

[IA2] V1 用基本命令による 2 ワード加算，減算，比較演算（1A，1B，1C 命令を使用）

1) 2 ワード加算では，下位ワード加算前に第 1 オペランドを他の GR ワードに退避しておき，加算後に第 1 オペランド（加算結果）と退避ワードを比較して，加算結果が小なら上位ワードへのキャリーありと判定して上位ワードの第 1 オペランドに 0001h を加算する.

2) 2 ワード減算では，下位ワード減算前に第 1 オペランドと第 2 オペランドを比較し，第 1 オペランドが小なら上位ワードからのボローありと判定して上位ワードの第 1 オペランドから 0001h を減算する.

3) 2 ワード比較では, 上位ワードから先に比較し, 結果が(=)の場合のみ下位ワード比較を行う.

[IA3] V1 用基本命令による 2 ワード連結 1 ビットシフト処理

1) 2 ワード連結 1 ビット上位シフトでは，上位ワードシフトを先に行ってから下位ワードシフトを行い, 下位ワードシフト結果での Overflow_bit (CC)が 1 の場合は上位ワードシフト結果の LSB へ 0001h を OR する.

2) 2 ワード連結 1 ビット下位上位シフトでは，下位ワードシフトを先に行ってから上位ワードシフトを行い，上位ワードシフト結果での Underflow_bit (CC)が 1 の場合は下位ワードシフト結果の MSB へ 8000h を OR する.

[1A4] V1 用 2 進加算命令（1A 命令）による 10 進加算処理［参考］

最大 4 桁の 2 つの 10 進数値について，下位ニブルから順にニブル単位で桁上がりを含めて 2 進加算し，結果が 9h を超えたら 6h を加算して 10 進補正する．（オーバーフロー検出なし）
ニブル補正用の数値 000Fh，0009h，0006h の上位ニブルへのシフトは 1 ビット上位シフト命令(22 命令)の 4 回繰り返しで行う.

[IA5] V1 用 2 進減算命令（1B 命令）による 10 進減算処理［参考］

Op2 の 10 進数値を 9999h から 2 進減算し 9 の補数に変換する．この演算ではボローは発生しない．この 9 の補数に 0001h を 2 進加算して 10 の補数にする．以降は上記の方法で 10 進加算する．（オーバーフロー検出なし）

◎ Inst_Advance シート理解度確認問題　　次の問いに答えよ.

(1) メインメモリの連続番地のワードを繰り返しアクセスするにはどのような手順が必要か.
　[IA1]の手順は，7 命令，9 命令以外のすべての RM 形式命令に適用できる.

(2) 2 ワード加算処理で，下位ワード加算での桁上げ（キャリー）の有無を判定するにはどのような手順が必要か.
　（[IA2] 1)　参照）

(3) 2 ワード減算処理で，下位ワード減算で上位ワードからの借り（ボロー）が必要か否かを判定するにはどのような手順が必要か.
　（[IA2] 2)　参照）

(4) 2 ワード比較処理はどのような手順で行うのか.
　（[IA2] 3)　参照）

(5) 2 ワード連結 1 ビット上位シフト処理（または同下位シフト処理）はどのような手順で行うのか.
　（[IA3]　参照）

(6) [参考] 2 進加算命令による 10 進加算はどのような手順で行うのか.
　（[IA4]　参照）

(7) [参考] 2 進加算命令による 10 進減算はどのような手順で行うのか.
　（[IA5]　参照）

(8) [参考] 2 進加算命令による 10 進加算および 10 進減算に必要なステップ数(Step Count)を，Calc_Prog シートの電卓処理（10 進／2 進変換使用）の場合と同じ入力値で比較してみよ.

6.6.1 (PE1) 基本的な数値計算プログラム(1ワード演算)　　Prog_Exercise1 シート

以下，例題と問題の項目と簡単な解説のみを示す．具体的なプログラムはそれぞれのシートを参照.

以下の**例題 PE1.1** と**問題 PE1.1** では, 数値は 2 バイト符号なし絶対値形式とし, 桁あふれ(オーバーフロー）は考慮しない.

例題 PE1.1　次の(1)～(4)のプログラムを書きなさい

(1)　GR#A と GR#B の内容交換
　一方の GR ワード内容を他の GR ワードに退避させてから上書きする.

(2)　GR#E ← GR#A×2－GR#B
　2 のベキ乗の乗算は加算命令で行える.
　GR ワードを 1 ビット上位シフトしても×2 になる.

(3)　[F0h] ← [0Ch]×4　　（番地による数値ワード指定）
　同上

(4)　[11h]／4 を計算し，商を[F0h]に剰余（余り）を[F1h]にストアする
　商は下位シフト，剰余は被除数へのマスク処理で求める.

問題 PE1.1　次の(1)～(5)のプログラムを書きなさい

(1)　[F0h]と[F1h]の内容を交換する
[F0h]と[F1h]は MM の RAM 領域のワードのため，準備としてサンプルデータをストアしておく．

(2)　GR#A ← (GR#B＋GR#C)×2－GR#D×3
×3 の計算は，式全体を×2 で計算し，さらに被乗数を 1 回減算して行う．
または，×2 の項を計算してから 3 回減算してもよい．

(3)　[F0h] ← [0Ch]＋[0Dh]＋[0Eh]　　（番地による数値ワード指定）
第 1 項を GR ワードにロードしてから残りの項をレジスタ・メモリ加算する．

(4)　[F0h] ← [0Ch]×10　　（番地による数値ワード指定）
×2 まで計算したら GR ワードに退避しておき，×8 を計算したらそれに加算する．

(5)　[11h]／8 を計算し，商を[F0h]に剰余（余り）を[F1h]にストアする
例題 PE1.1 (4) のシフトビット数とマスク処理のビット数を増やす．

6.6.2 (PE1)　条件分岐命令とその応用　　Prog_Exercise1 シート

以降の問題では，特に指定がない場合は，数値は 1 ワード（2 バイト）2 の補数形式（正／負あり）とする．また，桁あふれ（オーバーフロー）は考慮しない．

問題 PE1.2

次の 1)～13)のアセンブラ表記の演算をそれぞれ独立に実行するプログラムを書き，実行後のコンディションコード CC の値を調べて書きなさい．

ただし実行前のレジスタの値は GR#A=1373h，GR#B=BABAh とする

1)　AR A, A　　CC=
2)　AR A, B　　CC=
3)　SR A, A　　CC=
4)　SR A, B　　CC=
5)　CR A, A　　CC=
6)　CR A, B　　CC=
7)　SU 0, A　　CC=
8)　SU 1, B　　CC=
9)　SD 0, A　　CC=
10)　SD 1, B　　CC=
11)　A　B, 7B　　CC=
12)　S　B, 7B　　CC=
13)　C　B, 7B　　CC=

例題 PE1.2　次のプログラムを書きなさい

[F0h] ← max([0Dh], [0Eh], [0Fh])　（max は最大値を意味する．ここでは数値は符号なし絶対値形式とする）

第１項を GR ワードにロードし，残りの項と比較して大なら上書きする．

問題 PE1.3　次の(1)～(4)のプログラムを書きなさい

(1)　[F0h] ← max([0Dh], [0Eh], [0Fh])　（数値は負の場合を含む２の補数形式）

負の場合を含む２の補数形式の数値の大小比較は，正数は MSB を'1'に負数は MSB を'0'にしてから比較命令 1C（または C）を用いる．これには数値に 8000h を加算してから比較すればよい．元の値に戻すには再度 8000h を加算する．

(2)　[F0h] ← ｜[11h]－[0Ch]｜　（差を求め，負数の場合は絶対値に変換する）

負数判定は差を求める演算（減算）の CC で行う．

(3)　[F0h] ← ｜[11h]｜　（負数の場合は絶対値に変換する）

加減算結果ではないデータワードの負数判定は１ビット上位シフト（元のデータは保存）で行う．

(4)　GR#A の値が奇数ならそのまま，偶数なら２で割って[F0h]にストアする

奇数判定のための LSB の CC セットは１ビット下位シフト（元のデータは保存）で行う．

(5)　GR#A≠GR#B のとき，大きい方を[F0h]にストアする（負数の場合あり，GR#A，GR#B の内容は保存する．＝ のときはストアしない）

◎　Prog_Exercise1 シート理解度確認問題　次の問いに答えよ．

(1)　２つの GR ワードの内容を交換するにはどのような手順が必要か．

（**例題 PE1.1(1)** 参照）

(2)　メインメモリの RAM 領域の２つのワードの内容を交換するにはどのような手続が必要か．

RAM 領域の２つのワードを GR ワードへロードし，それぞれ元とは異なる RAM 領域ワードへストアする．（**問題 PE1.1(1)** 参照）

(3)　乗数が 2 のベキ乗の乗算（×2，×4，×8 など）は１ビット上位シフトを繰り返すことで計算できるが，乗数が 2 のベキ乗でない場合の乗算（×3，×5，×6 など）はどのようにすれば計算できるか考察せよ．

2 のベキ乗の乗算は，第１オペランドと第２オペランドの GR ワードを同じにしたレジスタ加算命令（GR ワードの値を２倍にする）を繰り返すことでも行える．×5 は元の値を５倍することなので，元の値を GR ワードに退避しておき，4 倍の値を計算してから元の値を１回加算すればよい．（**問題 PE1.1(2)** 参照）

(4)　除数が 2 のベキ乗の除算（／2，／4，／8 など）で剰余（余り）を求める方法を述べよ（注：／は÷の意味）．

2 のベキ乗の除算（／2，／4，／8 など）の商は，23 命令「1 ビット下位シフト命令」を繰り返すことで計算できる．剰余はその際に Under flow するビット列で与えられる．ビッ

ト列を合成する手続きは手間がかかるので，予め被除数を GR ワードに退避しておき，シフト回数分の'1'を LSB から並べた 2 進数マスク値（／8 では 3 回シフトするので 0000 0000 0000 0111b=0007h）を GR ワードに設定し，退避してある被除数にマスク値を 2A 命令で AND して剰余を得る．（例題 PE1.1 (4)，**問題 PE1.1(5)** 参照）

(5) 除数が 2 のベキ乗の除算（／2，／4，／8 など）の商は，1 ビット下位シフトを繰り返すことで計算できるが，除数が 2 のベキ乗でない場合の除算（／3，／5，／6 など）の商はどのようにすれば計算できるか考察せよ．

除数が 2 のベキ乗でない除算（／3，／5，／6 など）の商は乗算とは異なって簡単な操作では得られない．除算サブルーチン（Div_Prog シート参照）を用いる．

(6) **CC レジスタに演算結果のコンディションコードを設定する命令をすべて挙げよ．また，それぞれの命令で，どのような結果のとき CC にどのような値が設定されるのか説明せよ．**

1) A 命令「加算命令」，1A 命令「レジスタ加算命令」，B 命令「減算命令」，1B 命令「レジスタ減算命令」では，計算結果がゼロ（CC=0），負（CC=1），正（CC=2），オーバーフロー（CC=3）．

2) C 命令「比較命令」，1C 命令「レジスタ比較命令」では，第 1 オペランドと第 2 オペランドが＝(CC=0)，＜（CC=1），＞（CC=2）．

3) 22 命令「1 ビット上位(左)シフト命令」では，Overflow bit が '0' (CC=0)，'1' (CC=1)．
23 命令「1 ビット下位(右左)シフト命令」では，Underflow bit が '0' (CC=0)，'1' (CC=1)．

（以上 **問題 PE1.2** 参照）

(7) 複数の符号なし絶対値形式の 1 ワード数値データの中から最大値を求める手順を説明せよ．
（例題 PE1.2 参照）

(8) 2 の補数形式の 1 ワード数値データの正／負を判定する手順を説明せよ．
判定対象データが加減算結果のときはその CC，それ以外のときは，1 ビット上位シフト（元のデータは保存する）で CC 設定する．（**問題 PE1.3 (2) (3)**参照）．

(9) 2 の補数形式で正または負が混在している複数の 1 ワード数値データの中から最大値を求める手順を説明せよ．
（**問題 PE1.3 (1)** 参照）．

6.6.3 (PE2)　繰り返し処理（ループ）とその応用　　Prog_Exercise2 シート

例題 PE2.1　　2 進数 16 進表記で 1 から A までを加算し[F0h]にストアする

問題 PE2.1　次の(1)～(3)のプログラムを書きなさい

(1) GR#0 の上位バイトと下位バイトをそれぞれ GR#1，GR#2 の下位バイトに分けて格納する

(2)　1 バイトデータ 57h に含まれる 2 値の'1'の数を計数し[F0h]にストアする

(3)　　数値 5h の値を二乗し[F0h]にストアする

6.6.4 (PE2)　メインメモリへの連続アクセス　　Prog_Exercise2 シート

V1plus は，この目的のためにレジスタ間接メモリアクセス命令（38 命令，3D 命令）を備えているが V1 にはない．ここでは，それに替わる以下の方法でこの機能を実現する．

例題 PE2.2 (1)　0Ah 番地からの 5 ワードの和を求め [F0h]にストアする

GR ワードにレジスタ・メモリ加算命令を設定してメモリアクセスの番地 y2 を歩進しつつ MM-RAM 領域にコピーして実行する．MM-RAM 領域の加算命令の後にはプログラムへの復帰用無条件分岐命令をおく．

例題 PE2.2 (2)　ラベル vs からの 5 ワードの和を求め RAM 領域の[x0]にストアする

例題 PE2.2 (1) のすべての処理をラベルで行う

問題 PE2.2　次の(1)～(2)のプログラムを書きなさい

(1)　ラベル vs からの 6 ワードから正の最大値を選んで[x0]にストアする

(2)　ラベル vn からの 8 ワードから数値が奇数のワード数を求め[x0]にストアする

◎ Prog_Exercise2 シート理解度確認問題　　次の問いに答えよ．

(1) 繰り返し処理（ループ処理）において，ある条件を満足するまで処理を繰り返す手順を述べよ．

GR ワードに繰り返し処理の最終値を設定しておき，繰り返し処理の途中経過の値と最終値を比較し，分岐命令で=<の間はループを続ける．（**例題 PE2.1** 参照）

(2) 1 ワードデータにおいて，必要なビット以外の 2 値を'0'にする手順を述べよ．

必要なビットを'1'に他を'0'にしたマスク値のワードを用意し，2A 命令「論理積命令」で AND をとる．（**問題 PE2.1 (1)** 参照）

(3) 1 ワードデータにおいて，特定の連続したビットの値を，他の 1 ワードデータの対応するビットの値で置換する手順を述べよ（注：この処理は上記(2)の組み合わせで可能）．

一方のデータワードに上記(2)の操作を行い，そのマスク値ワードの'0'と'1'を反転したものを 21 命令「論理否定命令」で作成し，他方のデータワードのマスク処理を行う．先のマスク処理済みデータワードに後のマスク処理済みデータワードを 20 命令「論理和命令」で OR する．

(4) メインメモリの連続番地のワードを繰り返しアクセスする処理をラベルによる番地指定のみで行うには，番地を歩進（+1）する命令ワードに対して繰り返し処理開始前にどのような手順が必要か．

繰り返しアクセスを行う命令ワードの第 2 オペランドの y2 を 00h にしたものを GR ワードに用意する．最初のアクセス先のラベルを第 2 オペランドに書いた 9 命令「アドレスロード命令」でラベルを番地に変換したワードを GR に用意する．命令ワードに番地のワードを OR する．（**例題 PE2.2 (2)** 参照）

6.6.5 (PE3) 2ワード演算（サブルーチンの応用）　Prog_Exercise3 シート

V1plus は，下位ワード加算／減算での桁上げ（キャリー）を加える上位ワード用加算／減算命令（4A 命令，4B 命令）を備えているが V1 にはない．ここでは，下位ワード加算命令と比較命令で同様な機能を実現する．また，同様に 2 ワード連結シフトを実現する．

例題 PE3.1　次の(1)～(2)のプログラムを書きなさい

(1)　[F1h][F0h] ← [05h][04h]＋[07h][06h]　（2 ワード加算）

GR#A と GR#6，GR#B と GR#7 をペアレジスタとして用いる（後者が上位ワード用）

2 ワード加算用サブルーチンを作成する．（**[IA2] 1)** 参照）

(2)　[F1h][F0h] ← [07h][06h]／4　（2 ワード除算，小数点以下切捨て）

2 ワード連結 n ビット下位シフト用サブルーチンを作成する．（**[IA3] 2)** 参照）

問題 PE3.1　次の(1)～(2)のプログラムを書きなさい

(1)　[F1h][F0h] ← [05h][04h]－[07h][06h]　（2 ワード減算）

下位ワード減算で Op1<Op2 のときは Op1 上位ワードからボロー（借り）が発生するため前もって－1 する．サブルーチンは使用しない．（**[IA2] 2)** 参照）

(2)　[07h]の値を 2 ワード演算で 1024 倍し [F1h][F0h]にストアする

2 ワード連結 n ビット上位シフト用サブルーチンを作成する．（**[IA3] 1)** 参照）

◎　**Prog_Exercise3 シート理解度確認問題**　　次の問いに答えよ．

このシートの問いは Inst_Advance シートの(2)～(5)の問いと同じ．

(2)　2 ワード加算処理で，下位ワード加算での桁上げ（キャリー）の有無を判定するにはどのような手順が必要か．

（**[IA2] 1)，例題 PE3.1 (1)** 参照）

(3)　2 ワード減算処理で，下位ワード減算で上位ワードからの借り（ボロー）が必要か否かを判定するにはどのような手順が必要か．

（**[IA2] 2)，問題 PE3.1 (1)**　参照）

(5)　2 ワード比較処理はどのような手順で行うのか．

（**[IA2] 3)**　参照）

(5)　2 ワード連結 1 ビット上位シフト処理(または同下位シフト処理)はどのような手順で行うのか．

（**[IA3]，例題 PE3.1 (2)，問題 PE3.1 (2)** 参照）

6.6.6 (PE4)　複合問題　　Prog_Exercise4 シート

例題 PE4.1　次の(1)～(2)のプログラムを書きなさい

(1)　数値 63h の平方根を超えない最大の整数を求め [F0h]にストアする

以下の行順と数値の和の関係を利用する　（63h = 99d）

$1^2 = 1$

$2^2 = 1 + 3$
$3^2 = 1 + 3 + 5$

(2) ラベル vs から始まるワード列において，オールゼロのワードまでの和を上位ワードへの桁上げを考慮した２ワード演算で求め，RAM 領域の[x1][x0]にストアする．数値は符号なし絶対値形式．（無限ループを避けるため調べるワード数を最大 10 とすること）
問題 PE2.2 (2) の連続アクセスロードと**例題 PE3.1 (1)**の２ワード加算サブルーチンを変形して用いる．

問題 PE4.1　次の(1)～(3)のプログラムを書きなさい

(1) ラベル vs から始まるワード列において，最下位ニブルが 4h の最初のワード以前のワード数を求め，RAM 領域の[x0]にストアする（無限ループを避けるため調べるワード数を最大 10 とすること．また，10 ワード目までに見つからない場合はワード数をゼロとすること）

(2) 多倍長演算に関する問題：ラベル vs から始まる 5 ワードとラベル vn から始まる 5 ワードとを 5 ワード長で加算し，結果を RAM 領域のラベル x0 から始まる 5 ワードにストアする．なお，各 5 ワードで番地が小さい方を下位ワードとする（Little Endian）．下位ワード加算で生じるキャリー（桁上げ）を次のワードに加算する機能を設けること．ワード長を任意の n（≧１）で指定できるようにすること．
次キャリー発生は，キャリー加算，Op2 加算の双方で起こりうる点に注意．

(3) 乱数を扱う問題：ジャンケンゲーム
１バイトの乱数を GR ワードにロードして 3h で割り，剰余（余り）の 0h，1h，2h をグー，パー，チョキに対応させる．GR#0 にユーザ出し手をプログラム待機停止期間に入力し，機械の出し手と勝ち負け判定を行う．９回のジャンケンでのスコアを表示する．

◎ **Prog_Exercise4 シート理解度確認問題**　次の問いに答えよ．

(1) ある数値の平方根を超えない最大の整数を求める処理を，乗算サブルーチンまたは問題 PE2.1 (3)の方法を使用して行う手順を説明せよ．
数値 2h を初期値としてその値の二乗を求める操作を数値を歩進（+1）しつつ行い，二乗した値が目標値を超えたらその一つ前の二乗前の値を－1 して求める．

(2) あるデータワードの特定のニブルの値を調べる手順を説明せよ．
マスク処理で対象のニブルを取り出し比較命令と分岐命令で判定する．

(3) ２ワード以上の多倍長加算処理は，２ワード加算処理の場合と比較して，どのような追加の手順が必要か説明せよ．
２ワード以上の加算においてもキャリー加算（桁上げ処理）が必要．

(4) １バイトの乱数を 3h で割って得られる剰余（余り）が 0h，1h，2h になる確立が概ね等しいかどうか，各剰余が出現する「場合の数」を求めて考察せよ．
１バイトの乱数を 00h～FFh（0d～255d）とすると 00h を含む 3h の倍数は 85d 個，3 の倍数+1 は 84d 個，3 の倍数+2 は 84d 個となる．よって剰余が 0h になる確率は他より 1／85d 大きくなるが，これは無視できるレベルである．

6.7　プロセッサハードウェアでの CPU 命令サイクルとシーケンス制御（参考）

プロセッサで実行する機械命令プログラムでは，命令ワードごとに次の６種類の動作からなる CPU 命令サイクルを実行する：(1) 命令読み出し（命令フェッチ），(2) 命令デコード，(3) データ読み出し（オペランドフェッチ），(4) 演算実行，(5) 演算結果格納，(6) 次命令アドレス決定．ただし (6) は，通常の命令では (4) の開始時に実行し，(4)と(5)の期間に(1)を平行に実行する．RR 形式命令では，１命令サイクルにおいてこれらの動作を４ステージで（図６），RM 形式命令では６ステージで（図７）実行する．

RR 形式命令では，１命令サイクルでのメインメモリへのアクセスは命令読み出しの１回のみであるが，RM 形式命令では命令読み出しとデータ読み出し／書き込みの２回実行する．命令サイクルの各ステージで実行する動作とタイミングは，命令レジスタ（IR）にフェッチした機械命令をシーケンサ（SEQ）で解読して各機能モジュールの動作を指定する制御信号を生成するとともに，外部入力クロック信号から生成する基本内部制御信号に同期させることで実行するタイミングを決めている．

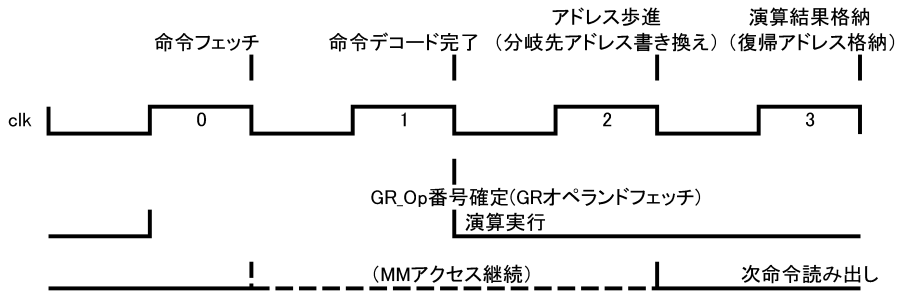


図６　RR 形式命令の命令サイクル（４ステージ）

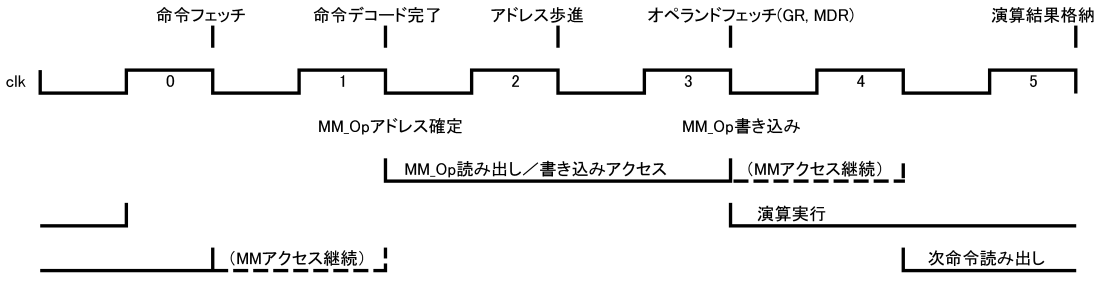


図７　RM 形式命令の命令サイクル（６ステージ）

図８　組込みプロセッサ開発用 FPGA ボード

- 三菱マイコンソフトウェア社製 FPGA 評価ボード MU200-EC12S（ALTERA 社製 Cyclone FPGA EC12S 搭載）.
- グラフィックス型論理回路設計システム Visual_Elite（Mentor Graphics 社製）で開発したプロセッサ論理ファイルを，Synopsys 社製論理回路最適化ソフト Synplify-pro でハードウェア記述言語 VHDL 記述に変換し，ALTERA 社製 FPGA 配置敗戦ソフト Quartus II で FPGA をプログラム（インプリメント）することでハードウェアプロセッサとして動作する.
- エミュレータでのプログラム実行速度は約 100 命令／秒であるが，FPGA では最速で約 1 千万命令／秒（10 MIPS，40MHz クロック使用時）で動作する（10 万倍高速）.
- Cyclone FPGA EC12S では，マルチコア仕様にした NT-Processor V1 プロセッサを 3 個までインプリメントできる.

第 7 章　プログラムの処理

この章では，プログラミング言語の種類，プログラミングにおけるプログラムの処理過程とツール類について学ぶ.

7.1 プログラミング言語

- 計算機が唯一理解できる言語は機械語（機械命令）であるが，人間にとってはわかり難くプログラミングには適さない．（注：NT-Processor V1は例外．機械命令でもプログラムを記述できる）
- プログラミングに適した人間のための言語　→ プログラミング言語
- プログラミング言語はアセンブラ言語(アセンブリ言語とも) とコンパイラ言語の2 種に大別される.

7.1.1 アセンブラ言語

・各命令が機械語命令に1：1 に対応
・計算機（CPU）によって異なる.
・アセンブル(assemble)：アセンブラ言語プログラムを機械語に翻訳すること
・アセンブラ(assembler)：計算機に翻訳するためのプログラム

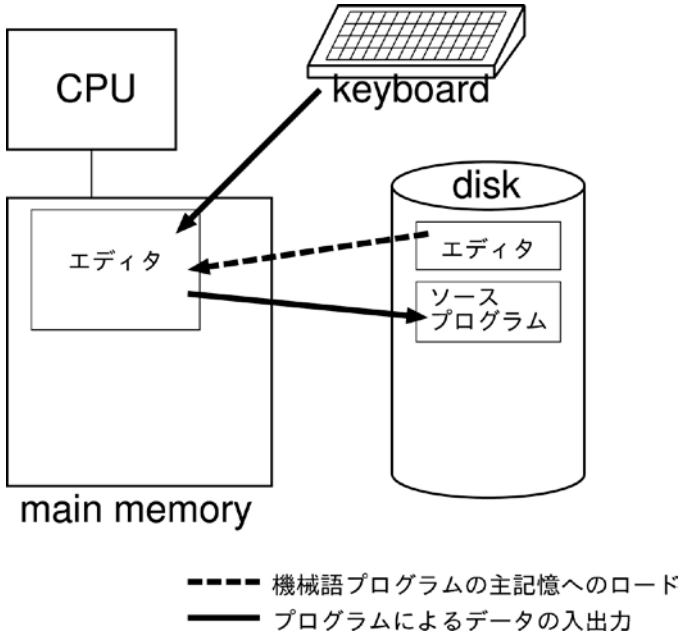
7.1.2 コンパイラ言語

・自然言語に近い形のプログラミング言語
・高級言語，高水準言語ともいう.
・一行が複数の機械語に対応する
・計算機には依存しない
・コンパイル(compile)：コンパイラ言語プログラムを機械語に変換すること
・コンパイラ(compiler)：その変換を行うための計算機プログラム

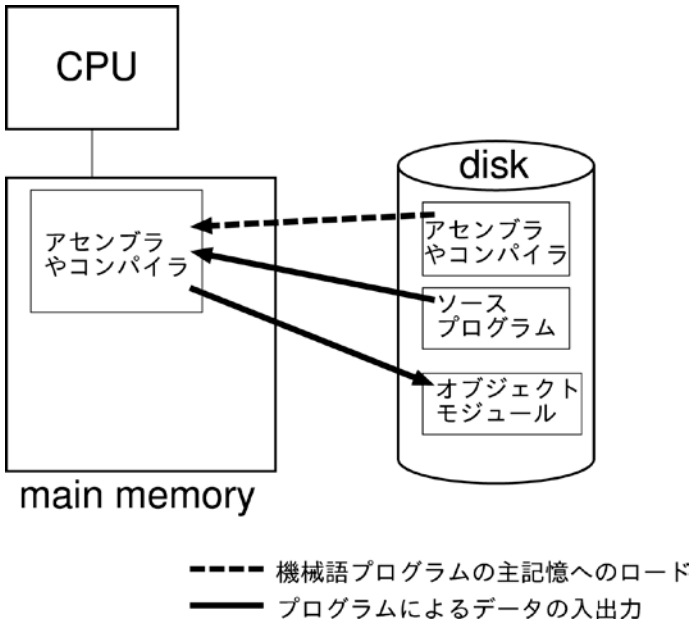
7.2 プログラムの処理過程

・プログラミング言語で書かれたプログラムをソースプログラム（source program）という．それを機械語に変換しないと実行できない.
・プログラムの作成から実行までは次の4 つの段階を経て行う
  1. ソースプログラム作成（テキストエディタを使う）
  2. 言語翻訳（アセンブラ，またはコンパイラを使う）
  3. 連係編集（リンカを使う）
  4. 実行

7.2.1 ソースプログラム作成

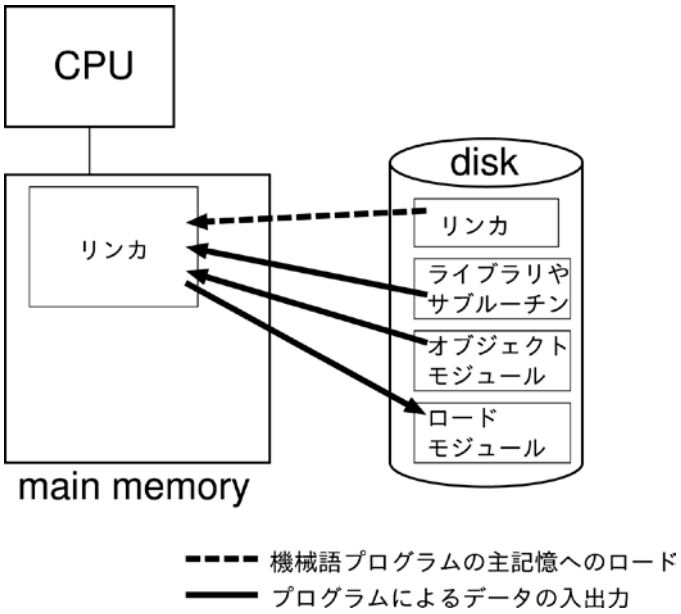


7.2.2 アセンブル(コンパイル)



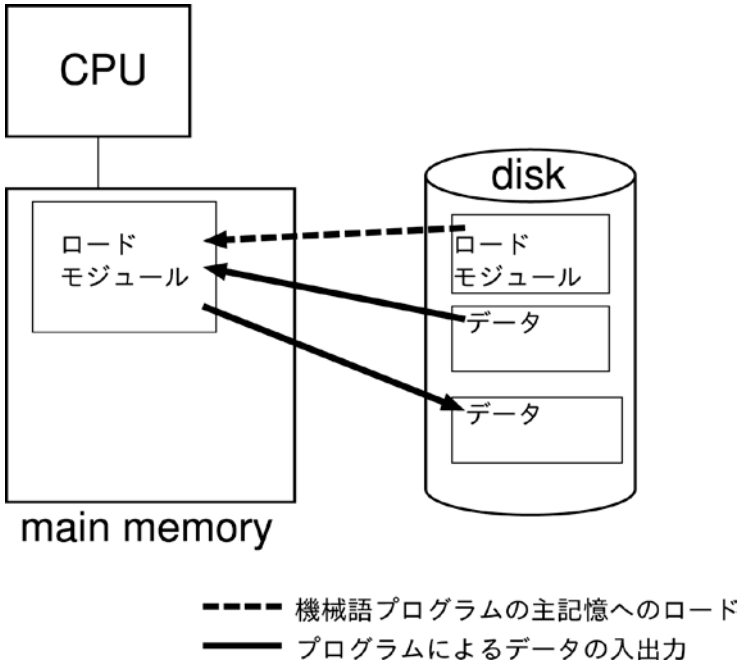
オブジェクトモジュール(object module)：機械語には変換されたが，外部関数などのサブルーチンがまだ組み込まれていないので実行可能でない．

7.2.3 連係編集



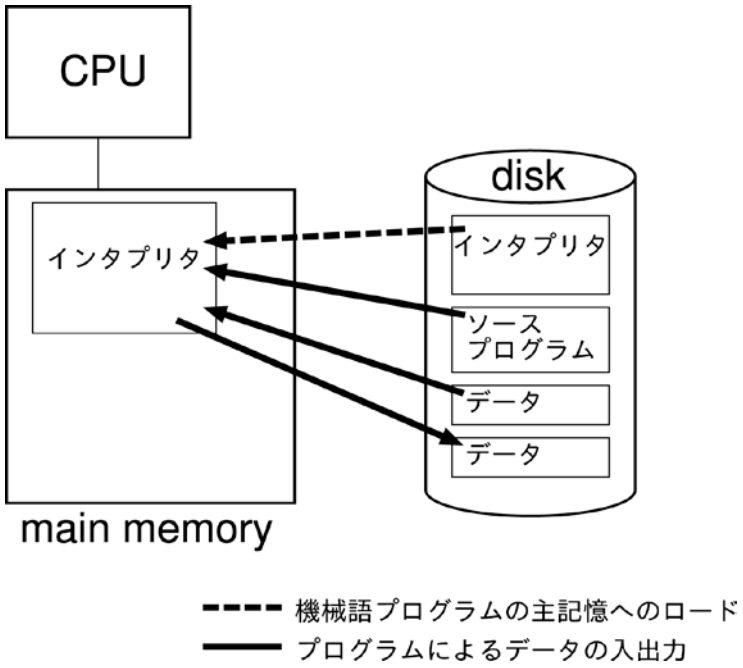
ロードモジュール(load module)：実行に必要なサブルーチンが組み込まれて実行可能になっている.

7.2.4 実行



7.3 インタプリタ(interpreter)

7.3.1 インタプリタによる処理



・プログラム実行時に，ソースプログラムの命令を解釈して機械語に変換して実行するためのもの.
・言語翻訳，連係編集，実行をまとめて一度に行うことができる.
・インタプリタによる実行方法を「通訳方式」ともいい，対して7.2 を「翻訳方式」ともいう.

7.3.2 コンパイラ方式との比較

長所

・多くの段階を経ないので簡便
・誤りを見つけやすい
・コンパイラより作りやすい

短所

・実行が遅い
・モジュール化に不向き
・単独実行不可．つまり，実行にはインタプリタが必要

7.4 統合プログラミング環境

・ソースプログラム作成から実行までの段階を（内部的には存在するが）意識することなく行うことのできるシステム.
・内部に各処理段階のプログラムを含んでいるか，自動的に呼び出して実行されている.
・代表的な製品名はVisaul Stdudio，Eclipse など.
・コンパイラ・インタプリタ双方を含むものもある.
・文字形式で入力された数値が自動的に変換されて，計算され，再び文字で出力する処理が自

動的に行われるのに似ている.

7.5 まとめ

・プログラミング言語で書かれたプログラムを実行するには多くの処理過程が必要.
・翻訳（コンパイラ）方式と通訳（インタプリタ）方式がありそれぞれ長所・短所がある.
・統合プログラミング環境を使用してもこれら一連の段階は内部で自動的に処理されている.

「以上」

第 6 章著者： 津田 伸生 教授
第 7 章著者： 津田 伸生 教授
宮田 英男 名誉教授
鷹合 大輔 准教授

教材の取り扱いについて