修士論文

プロセッサデータパス可視化の検討と実現

氏　　名　：　天井　康雄

学籍番号　：　6162110075-0

担当教員　：　山崎 勝弘 教授

提 出 日　：　2013 年 2 月 15 日

立命館大学大学院 理工学研究科 創造理工学専攻

内容梗概

本論文では，コンピュータの動作原理を理解するために，プロセッサデータパスの可視化方法について検討し，教育用ボードコンピュータ KUE-CHIP2 を対象としたデータパスの可視化システムについて述べる．

本可視化システムは，命令レベルとクロックレベルの 2 つの可視化を用意し，命令単位，クロック単位，それぞれでデータの流れを理解できるようにしている．複数の例題と，ユーザー記述プログラムに対して可視化の例を示している．本研究で提案するプロセッサデータパスの可視化システムは，コンピュータの知識が少ない人でも，データパスのことについて理解してもらえるものを目標に検討した．

実際に作成した可視化システムでは，データパス上に，演算上で変化した数値を表示することができたが，データの移動経路や制御信号も表示することを望まれる．

目次

1. はじめに 1
2. プロセッサデータパスの可視化 2
   2.1 プロセッサデータパス 2
   2.2 可視化の設計方針 3
3. データパスの可視化方法 6
   3.1 対象プロセッサ 6
   3.2 オブジェクトコードの解析 6
   3.3 データパスの表示 7
4. データパスの可視化の実現 9
   4.1 KUE-CHIP2 シミュレータの構成 9
   4.2 演算命令 11
   4.3 ロード・ストア命令 11
   4.4 分岐命令 13
5. 可視化の実行例と考察 15
   5.1 例題 15
   5.2 ユーザー記述プログラム 26
   5.3 考察 28
6. おわりに 30

謝辞 31

参考文献 32

図目次

図 1：データパス 2
図 2：PC の値に 4 加算するときのデータパス 2
図 3：システムの全体構成 3
図 4：命令レベルの可視化 4
図 5：クロックレベルの可視化 5
図 6：KUE-CHIP2 の全体ブロック構成 6
図 7：全体の処理 8
図 8：シミュレータ全体構成 9
図 9：情報保持クラスの構成 10
図 10：演算命令のフローチャート 11
図 11：ロード命令 12
図 12：ストア命令 12
図 13：分岐命令 14
図 14：2 値の加算プログラム - 実行 16
図 15：他の数値での実行結果 17
図 16：N 個の中の最大値 - 前半の処理過程 18
図 17：N 個の中の最大値 - 後半の処理過程 19
図 18：1 から N までの和 - 実行結果 21
図 19：ユークリッド互除法 - 実行結果 23
図 20：バブルソートによる整列 - 実行 25
図 21：データのコピーを行うプログラム - 実行過程と結果 27

表目次

表 1：KUE-CHIP2　命令一覧 7
表 2：ユーザープログラムの記述例 8
表 3：分岐条件 13
表 4：2 値の加算のプログラム 15
表 5：N 個の中の最大値のプログラム 17
表 6：1 から N までの和のプログラム 20
表 7：ユークリッドの互除法による最大公約数のプログラム 22
表 8：バブルソートによる整列のプログラム 26
表 9：80 番地の数値を 81 番地にコピーするプログラム 27

1. はじめに

コンピュータの発展の速度はとても速く，最近では，スマートフォンやタブレットコンピュータの分野の発展が著しい．スマートフォンやタブレットコンピュータのように，持ち運びが容易である小型のものでも，高性能・高機能化が進んでいる．また，それらは，誰でも使いやすいことを目的とした，ナチュナル・ユーザー・インターフェースで表現されるため，今までコンピュータに親しまなかった人でも，直感的に使うことができる．そのこともあり，コンピュータに触れる機会はとても増えてきている．

しかし，残念ながら基本的な動作原理を理解している人は，コンピュータを触れる人口に対し，極端に数少ない．また，自発的に勉強をするにも，専門書を知識もなく読むのはハードルが高いと感じる人も少なくない．しかし，前述のように，コンピュータに触れる機会が多くなっている現代社会においては，コンピュータの動作原理の理解がある方が望ましいのは至極当然であろう．また，理工学生においても，基本原理の理解の必要性もある．

このような背景から，本研究室では，動作原理の 1 つであるデータパスの学習システムの開発に取り組むこととなった．データパスとは，メモリレジスタや ALU といった，演算に必要な機能を持つ要素と，コンピュータが演算を行う上で，データがそれらの要素を通る道筋である．学習システムは，コンピュータの知識が少ない人にも，データパスがどのように動作しているのか理解してもらえることを目標に，開発を進めた．

本研究では，データパスの理解をしやすくするにはどのようにすれば良いのか，という観点から，学習書を読み考えるよりも，目で見て考える方が理解しやすいと我々は考え，データの推移を可視化するという方法を採った．可視化をするにあたって，学習用ボードコンピュータである KUE-CHIP2 と，それに対応した命令セットを対象にした．データパスの推移を表現するために，まずは KUE-CHIP2 シミュレータを，プログラミング言語である Java で作成した．そして，KUE-CHIP2 のブロック図と共に，シミュレータの結果をディスプレイに表示することで，データの変化を可視化する．また，データパスの可視化に伴う表現方法についても試行錯誤する．

第 2 章では，データパスのことについて，また，可視化についての説明する．第 3 章では，対象プロセッサと，データパスの可視化方法について説明する．第 4 章では，可視化したときに，どのように表示されるのか説明する．第 5 章では，例題を用いて，実際に可視化されるのか説明し，また，考察する．

2. プロセッサデータパスの可視化

2.1 プロセッサデータパス

コンピュータのハードウェアは，ALU やメモリといった，機能を持つモジュールが複数集まり成り立っている．プロセッサデータパスとは，コンピュータが演算処理を行うときに，データが演算に必要な機能モジュールを通る転送路のことである．一般的な，データパスを図 1 に示す．

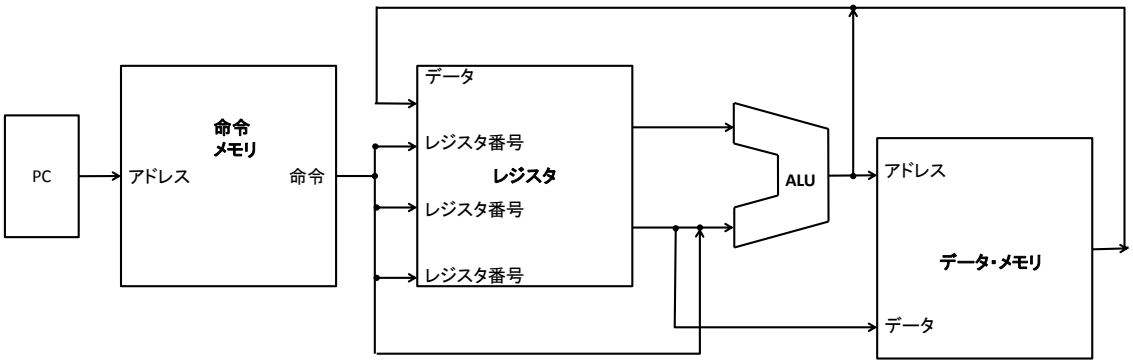


図 1：データパス

図 1 では，プログラムカウンタ(PC)が指し示すアドレスに格納された命令が，命令メモリから読み出され，レジスタ読み出し，ALU 演算，レジスタ書き込み，データメモリアクセスなどを行って，各命令を実行する．

例えば， PC の値を 4 つ繰り上げる場合のデータパスを，図 2 のように示す．

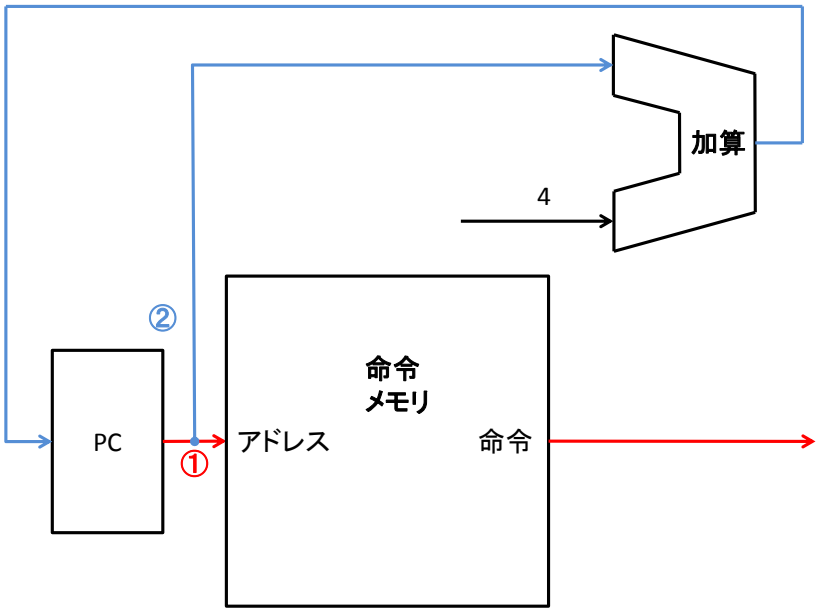


図 2：PC の値に 4 加算するときのデータパス

① PC の値が命令メモリに送られ，それをメモリ番地とする命令メモリの中身である数値が次のデータパスへ送られる．
② PC の値が加算 ALU に送られ，PC の値が 1 つ繰り上げられ，PC の中身が上書きされる．

2.2 可視化の設計方針

目に見えるようにすることによって，物事を把握しやすくできる．この利点から，学習効率を向上させることができる．作成するソフトウェアの設計方針は，以下の 3 つを軸にしている．可視化システムの全体構成を， 図 3 に示す．

- 命令レベルと，クロックレベルのデータパスの可視化を行う
- 例題のアニメーションを再生する
- ユーザーの記述プログラムをアニメーションにして再生する

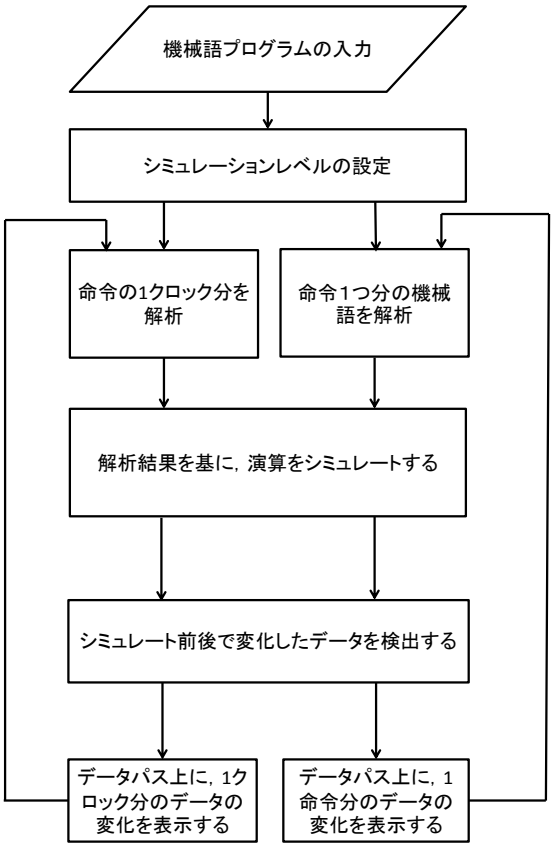


図 3：システムの全体構成

可視化システムは大きく別けて 2 つのプログラムから成り立っている． 1つは命令をシミュレートするシミュレートプログラム，そして，そのシミュレート結果の数値を画面上

に表示させる可視化プログラムである．可視化をするにあたって，まずは画面上に回路図を表示する．次に，KUE-CHIP2 シミュレータに，シミュレートする機械語プログラムと，プログラム実行に必要な数値を与える．そして，1 つの命令を実行したときの結果を表示する命令レベルと，1 つの命令に対して 1 クロックずつの結果を表示するクロックレベルの 2 種類のアニメーションを作成し，どの値が変わるのかを表示する．ソフトウェア中で，5 つの例題と，ユーザープログラムのデータパスの可視化を行えるようにする．

2.2.1　命令レベル

命令レベルのアニメーションでは，1 つの命令を実行したときに，実行前の情報がどのように変化するのがわかるようにする．例えば，ロード命令を行うとする．アセンブリ言語としては，LD ACC, 05 とする．その場合は図 4 のように表示させる．内部メモリ(Inter. Mem)から，05 という値が，ACC まで動き格納する様子を示す．

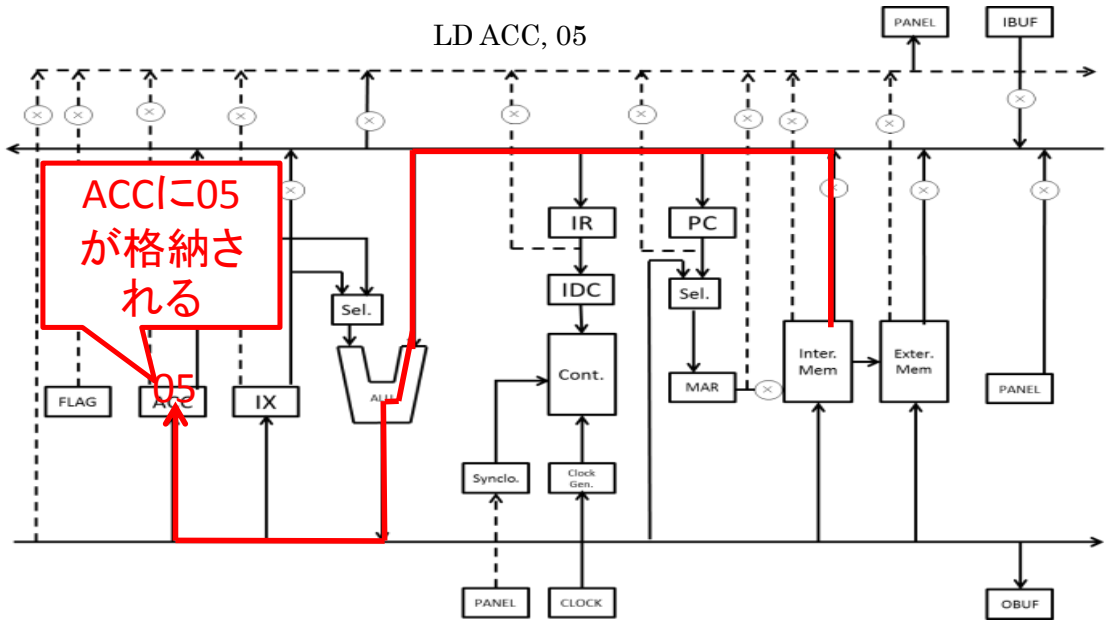


図 4：命令レベルの可視化

2.2.2　クロックレベル

クロックレベルアニメーションにおいては，1 クロックごとの情報の変化を順に表示させる．例として，10 進数の 10 という数値を，アキュムレータ(ACC)にロードする場合を説明する．アセンブリ言語としては，LD ACC, 10 とする．レジスタ直接アドレスのロード命令は，以下の(a)，(b)，(c)，(d)の 4 ステップで実行される．また，それらは図 5 のようになる．

(a)　プログラムカウンタ(PC)からメモリ・アドレス・レジスタ(MAR)へ値が転送される．

そして，PC の値である 00 に 1 が足され，01 になる．

(b)　MAR に格納された値 00 に該当する，メモリ番地 00 に格納された数値である 62 が命令デコーダ(IDC)に送り，何の命令かを解読する．

(c)　(a)と同様にして，　PC から MAR へ 01 という値が転送される．そして，PC の値である 01 に 1 が足され，02 になる．

(d)　MAR が指し示す番地 01 の中身である 05 が，算術演算装置(ALU)を通り，アキュムレータ（ACC)へ 05 が格納される．

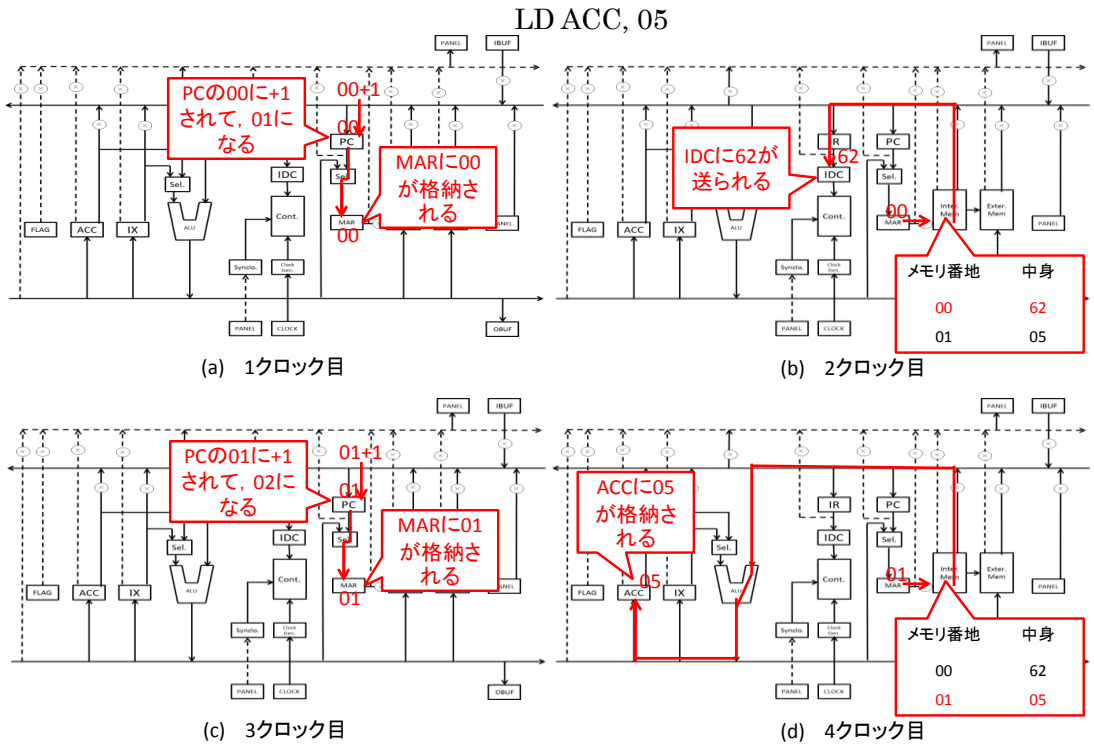


図 5：クロックレベルの可視化

3. データパスの可視化方法

3.1 対象プロセッサ

対象プロセッサは教育用ボードコンピュータ KUE-CHIP2 である．KUE-CHIP2 のブロック構成を図 6 に示す．

KUE-CHIP2 は 8 ビット・マイクロプロセッサである．命令は 19 種類，アドレス指定方式は 4 種類である．命令を実行する順序は，まずプログラムカウンタ(PC)の値が，メモリ・アドレス・レジスタ(MAR)に送られ，それが指すメモリ番地から命令番地が読みだされる．その命令は命令レジスタ(IR)に転送され，命令デコーダ(IDC)に転送される．こうして，ハードウェア各部の制御信号が生成される．

算術論理演算装置(ALU)は加減・論理・シフトなどの各種演算を実行する．ALU の左入力にはアキュムレータ(ACC)，あるいはインデックス・レジスタ(IX)から，右入力には ACC，IX，またはメモリから読み出されたデータが入力される．それらのデータに対して ALU で実行され，その結果が ACC か IX に格納される．

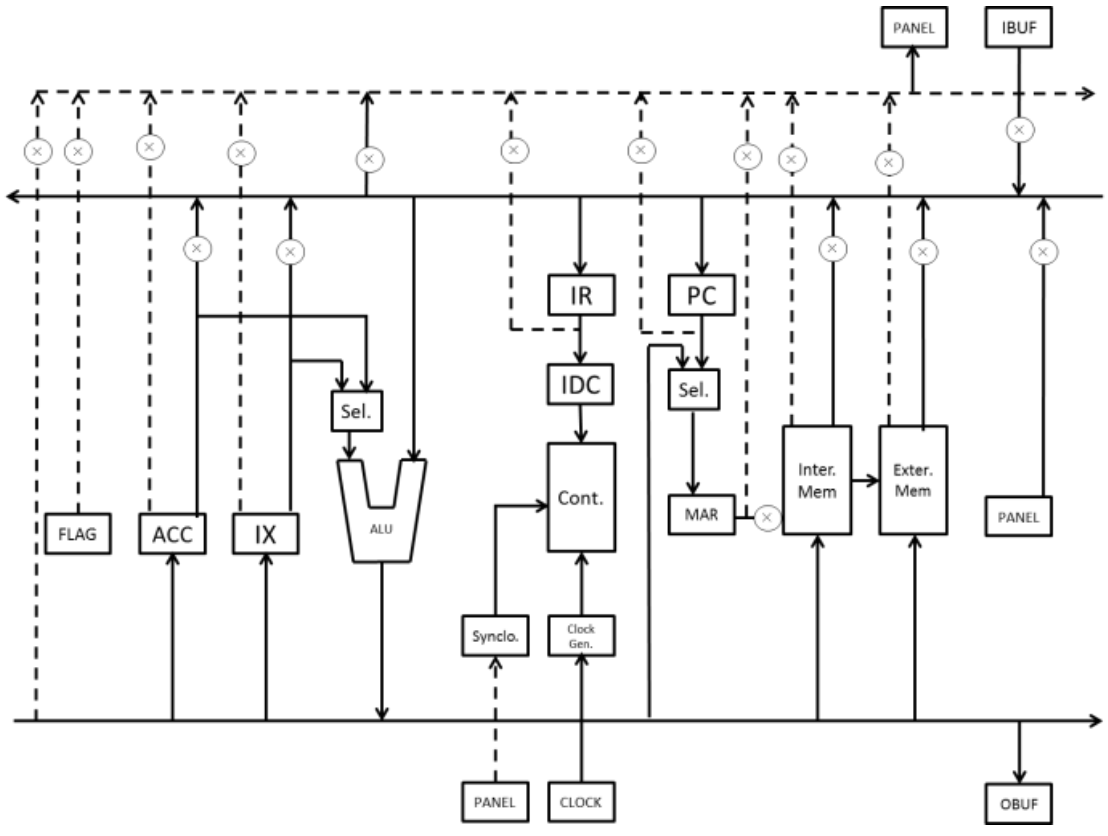


図 6：KUE-CHIP2 の全体ブロック構成

3.2 オブジェクトコードの解析

KUE-CHIP2 は，8 ビットで 19 種類の命令を表現する．演算命令，ロード・ストア命令

では計算先のメモリ・アドレスも含め，表現する．分岐命令においては，分岐条件も同時に表現する．KUE-CHIP2 の命令コードは表 1 のように表される．

表 1：KUE-CHIP2　命令一覧

略記号	命令コード(8 ビット)	意味
NOP	0 0 0 0 0 － － －	停止
HLT	0 0 0 0 1 － － －	停止
	0 1 0 1 － － － －	停止
OUT	0 0 0 1 0 － － －	ACC の値を外部に出力
IN	0 0 0 1 1 － － －	外部から ACC に入力
RCF	0 0 1 0 0 － － －	キャリアフラグを 0 にする
SCF	0 0 1 0 1 － － －	キャリアフラグを 1 にする
Bcc	0 0 1 1 － － － －	条件分岐
Ssm	0 1 0 0 － 0 － －	シフト命令
Rsm	0 1 0 0 － 1 － －	シフト命令
LD	0 1 1 0 － － － －	ロード命令
ST	0 1 1 1 － － － －	ストア命令
SBC	1 0 0 0 － － － －	キャリーフラグを用いて引き算
ADC	1 0 0 1 － － － －	キャリーフラグを用いて足し算
SUB	1 0 1 0 － － － －	引き算
ADD	1 0 1 1 － － － －	足し算
EOR	1 1 0 0 － － － －	EXOR
OR	1 1 0 1 － － － －	OR
AND	1 1 1 0 － － － －	AND
CMP	1 1 1 1 － － － －	比較（引き算）

表 1 のように，上位 4 ビットでほとんどの命令を区別することができる．NOP と HLT，OUT と IN，RCF と SCF の場合は 4 ビット目が 0 か 1 かで区別をする．Ssm と Rsm は 3 ビット目で区別をする．ハイフンで書かれた箇所のビットで，使用するレジスタや分岐条件を表現する．

3.3　データパスの表示

可視化をソフトウェアで実現するためには，プログラムを実行したときに，まず命令の種類は何か，演算命令であれば，どのレジスタを使用するのか，分岐命令であれば，どのような分岐条件なのか，というような情報が必要不可欠である．そのため，以下の図 7 のように各命令をシミュレーション処理させる．

まずは，ソフトウェア側であらかじめ用意したプログラム，あるいはユーザーが作成した機械語プログラムを読み込む. ユーザー記述プログラムの書式としては, 表 2 のように, 1 行に 1 命令を記述したものを想定している．1 つの命令に 2 語使うものは，1 語目と 2 語目の間にスペースを入れて区切るようにして記述する.

そして，プログラムを 1 行ずつ読み込み，16 進数の文字列を 2 進数に変換し，命令の種類やアドレッシングモードを判別させる. その結果を用いて, 各命令をシミュレートする. この処理を，最後の行に辿り着くか，あるいは HLT 命令を処理するまで繰り返し行う.

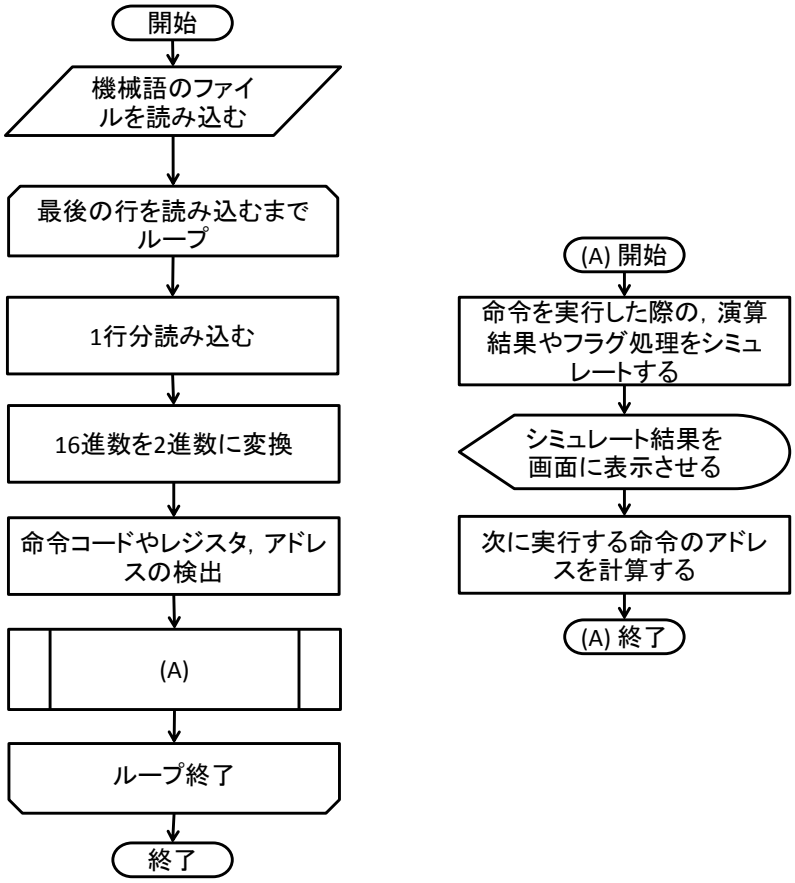


図 7：全体の処理

表 2：ユーザープログラムの記述例

行数	1 語目	2 語目	意味
1	64	80	80 番地の中身を ACC に格納する
2	b4	81	ACC の値と, 81 番地の値を加算する
3	74	82	82 番地に計算結果を格納する
4	0f		プログラム終了

4. データパスの可視化の実現

4.1 KUE-CHIP2 シミュレータの構成

KUE-CHIP2 シミュレータの構成は，レジスタやフラグ，メモリの情報を保持する情報保持クラスと，その情報を基に演算する命令演算を行う命令実行クラス，そして，その 2 つのクラスを管理・制御するクラスで構成されている． シミュレータの全体の構成を図 8 に示す.

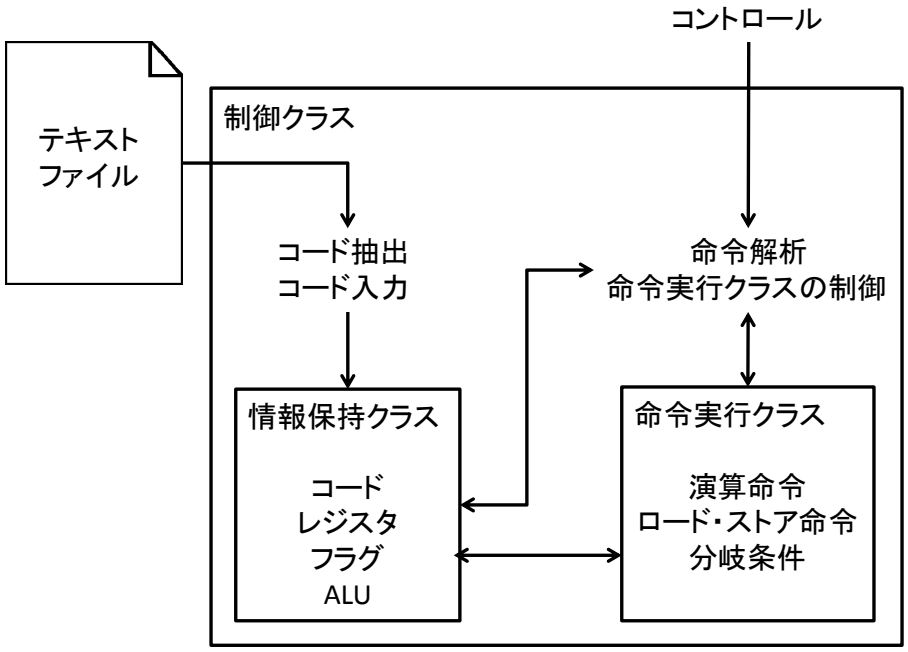


図 8：シミュレータ全体構成

シミュレータは，制御クラスがテキストファイルから，コードを抽出し，情報保持クラスに命令コードなどの情報を入力する．そして，制御クラスはその情報を基に，命令を解析し，命令実行クラスのメソッドに情報を渡す．命令実行クラスは受け取った情報を基に，命令を詳細に解析する．また，命令実行に必要な情報を，情報保持クラスから受けとる．そして，命令を実行する．

4.1.1 情報保持クラス

情報保持クラスでは，1 つの番地分のコード，ACC，IX のレジスタデータ，ALU の演算結果，フラグのデータの 4 種類の情報を記録する．命令コードの情報は 1 語目であるか，あるいは 2 語目であるかを区別させる．

このクラスを一次元配列型で宣言することによって，KUE-CHIP2 のメモリと同じように扱える．例えば，図 9 のように，情報保持クラスを構成要素数が 256 個の一次元配列として宣言する．KUE-CHIP2 の 00 番地の命令コードにアクセスする場合は，配列のインデ

ックス 0 が，KUE-CHIP2 の 0f 番地であれば，配列のインデックス 15 が相当する．もちろん，ff 番地であれば，配列のインデックスは 255 である．

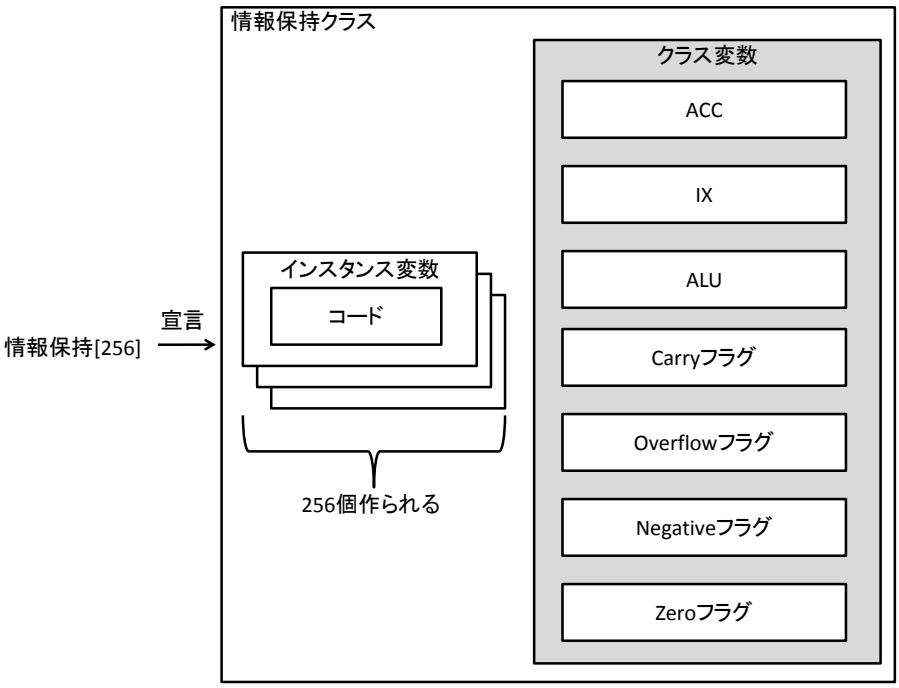


図 9：情報保持クラスの構成

4.1.2 制御クラス

制御クラスで行うことは，外部から受け取ったテキストファイルからコードの抽出，情報保持クラスにコードの入力，命令コードの解析，命令実行クラスの制御・実行，外部のプログラムの情報の受け渡しである．外部プログラムから KUE-CHIP2 をシミュレートする場合は，この制御クラスを介して実行することができる．今回作成した可視化システムも，可視化プログラムが制御クラスを介して，シミュレータ結果を取得し，それを表示している．

4.1.3 命令実行クラス

命令実行クラスでは，制御クラスから，実行する命令のためのメソッドを指定し，命令の実行に必要な 1 語目や 2 語目を渡され実行する．命令の種類に応じて，情報保持クラスからレジスタや，フラグなどを受け取り，KUE-CHIP2 の命令をシミュレートする．そして，演算結果やフラグ情報を情報保持クラスに渡す．分岐命令を実行した場合は，次に実行する命令の番地を制御クラスに返す．

4.2　演算命令

演算命令は算術演算系列の SBC, ADC, SUB, ADD と論理演算系列の EOR, OR, AND, 比較を行う CMP の 8 種類である. SBC と ADC は計算にキャリーフラグを追加し計算するものである．CMP は 2 つの数値を比較するものである．これらをクロックレベルで可視化するためには，図 10 のように処理していく．命令レベルでは，最後の可視化部分のみ動作するとする.

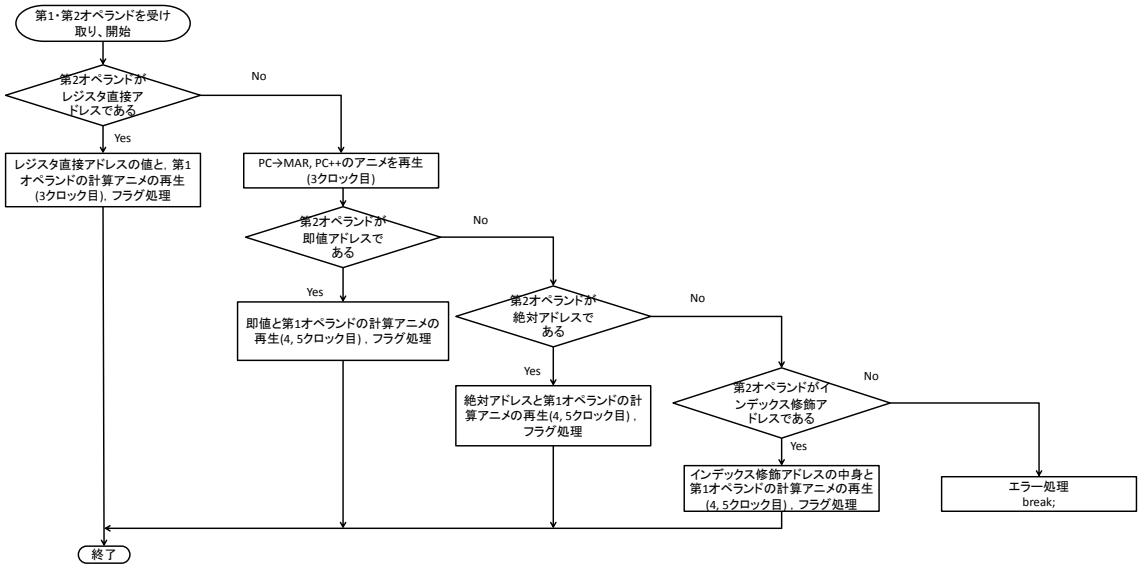


図 10：演算命令のフローチャート

演算命令では，図 7 の(A)において，実行する演算命令用のメソッドに第 1 オペランドと第 2 オペランドを渡し，処理を開始する.

まず，第 2 オペランドがレジスタ直接アドレスであれば，第 2 オペランドが指すレジスタの値と，第 1 オペランドの計算を可視化する．また，演算結果に基づいてフラグの処理も行う.

第 2 オペランドがレジスタ直接アドレスでなかった場合は，PC の値が MAR に入り，PC の値に 1 足される様子を可視化する．そして，第 2 オペランドが即値アドレスか，絶対アドレスか，インデックス修飾アドレスか，という条件に応じて処理を行っていく．また，演算結果に基づいてフラグの処理も行う.

第 2 オペランドがどの条件にも当てはまらないとき，エラー処理を行い，図 7 のループ中から抜け出す.

4.3　ロード・ストア命令

ロード命令，ストア命令は共通して，演算命令のようなフラグの処理はない.

ロード命令では，演算命令と同様のアドレッシングモードに対応しているため，処理中

の分岐条件も，演算命令と同じように処理していく．クロックレベルのロード命令可視化のフローチャートを図 11 に示す．演算命令と同様に，命令レベルでは，最後の可視化部分のみ動作するとする．

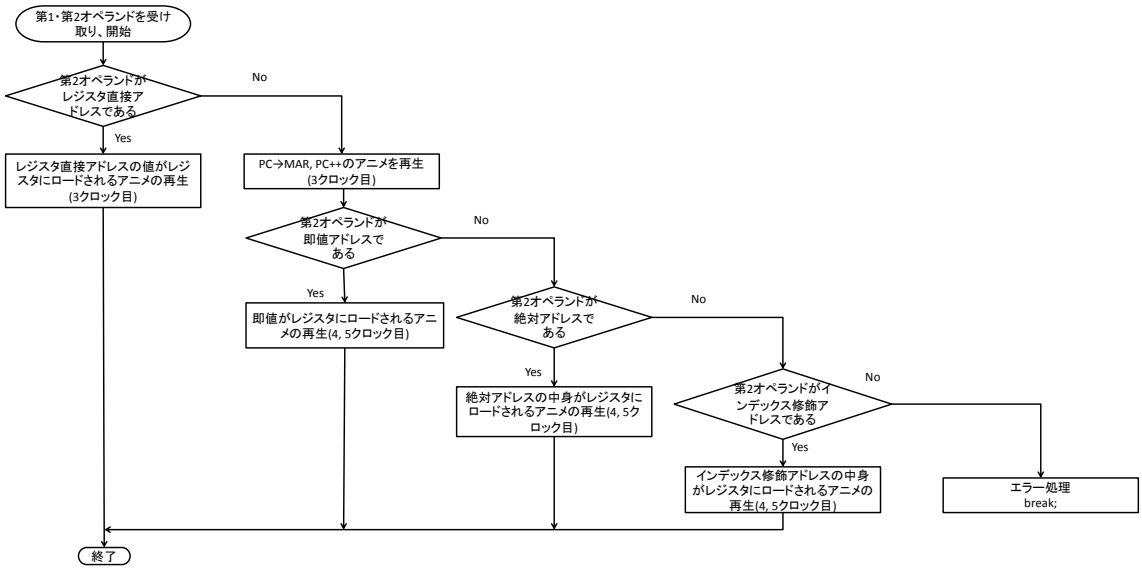


図 11：ロード命令

ストア命令では，ロード命令，演算命令よりも対応しているアドレッシングモードが少なく，絶対アドレスと，インデックス修飾アドレスのみなので，2 つの分岐条件で処理を行う．図 12 にストア命令のフローチャートを示す．命令レベルの可視化については，ロード命令と同様に，最後のかすか部分のみ動作するとする．

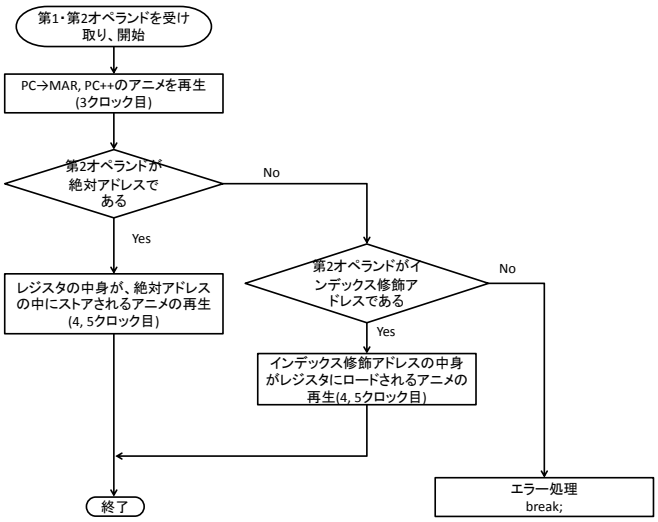


図 12：ストア命令

4.4　分岐命令

分岐命令の場合は，表 3 のように，1 語目の下位 4 ビットでどのような分岐条件か判別する．分岐条件は全 16 種類であり，A(常に実行)を除き，演算結果が桁上げしたか，オーバーフローしたか，負の数になるか，0 になるか，といった 4 種類のいずれかのフラグから判別し，実行する．図 13 にクロックレベルの分岐条件可視化のフローチャートを示す．演算命令，ロード・ストア命令と同様に，命令レベルでは，最後の可視化処理のみ動作するとする．

表 3：分岐条件

略記号	下位 4 ビット	実行条件
A	0 0 0 0	常に成立
VF	1 0 0 0	桁あふれ
NZ	0 0 0 1	0 でない
Z	1 0 0 1	0 である
ZP	0 0 1 0	0 以上
N	1 0 1 0	負の数のとき
P	0 0 1 1	正の数のとき
ZN	1 0 1 1	0 以下
NI	0 1 0 0	外部からの入力があるとき
NO	1 1 0 0	外部への出力があるとき
NC	0 1 0 1	桁上げがないとき
C	1 1 0 1	桁上げがあるとき
GE	0 1 1 0	A－B ≧ 0 のとき
LT	1 1 1 0	A－B＜0 のとき
GT	0 1 1 1	A－B＞0 のとき
LE	1 1 1 1	A－B ≦ 0 のとき

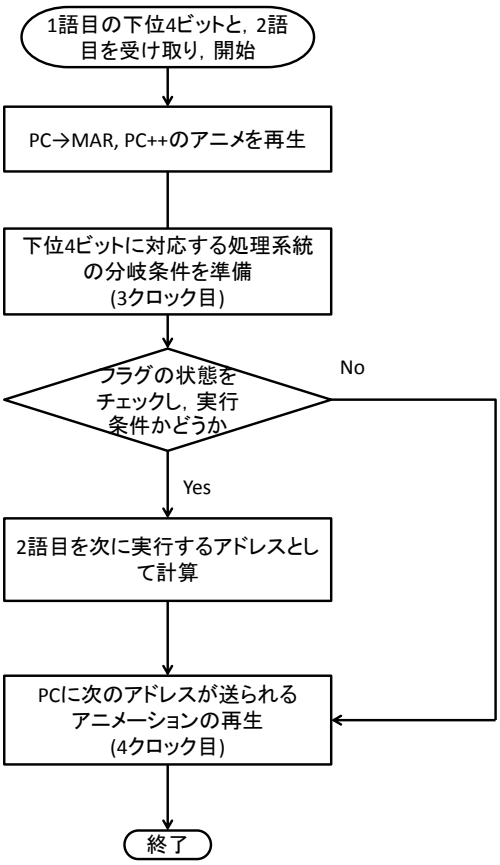

1語目の下位4ビットと，2語目を受け取り，開始
PC→MAR, PC++のアニメを再生
下位4ビットに対応する処理系統の分岐条件を準備
(3クロック目)
フラグの状態をチェックし，実行条件かどうか
No
Yes
2語目を次に実行するアドレスとして計算
PCに次のアドレスが送られるアニメーションの再生
(4クロック目)
終了

図 13：分岐命令

5. 可視化の実行例と考察

5.1 例題

5.1.1 2 値の加算

80 番地と 81 番地に入力した数字を加算し，81 番地に格納する．80 番地と 81 番地の値は任意で入力する．プログラムとしては，表 4 のようになる．アセンブリプログラム欄の[D1]は 80 番地，[D2]は 81 番地，[AND]は 82 番地を指す．また，実行結果を図 14 に示す.

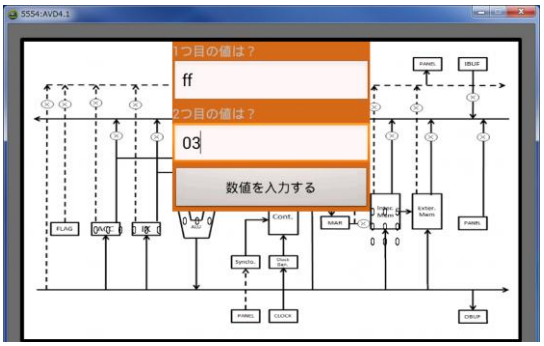
今回は-1 と 3 の加算を行う.図 14 の(a)において,16 進数でそれぞれの数値を入力した.次に，(b)では，-1 と 3 が入力されたことが確認できる．(c)，(d)，(e)，(f)では，それぞれ順に命令を実行している様子である．(d)において，-1 + 2 が計算され，2 という数値が ACC に格納され，(e)の ST 命令で，計算結果の 2 が格納されたことがわかる．このことから，演算結果とその過程を可視化できた

図 14 では負の数と正の数という組み合わせで計算したが，図 15 に示す結果は，正の数同士，負の数同士，演算結果がオーバーフローする正の数同士，演算結果がオーバーフローする負の数同士のものである.

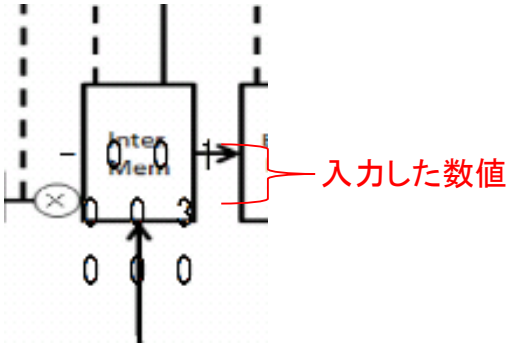
図 15 に表示される上から 2 つの数字が，加算する数値，下の数字が加算結果を Inter. Mem に格納したものである．(a)では，1 + 1 が実行され，結果の 2 が，(b)では，-1 + (-1)の結果である-2 が，(c)では，127 + 1 のオーバーフローの結果である-128 が，(d)では-1 + (-128)のオーバーフローの結果である 127 が格納された．この結果から，あらゆる組み合わせの加算が正常にシミュレートできたといえる.

表 4：2 値の加算のプログラム

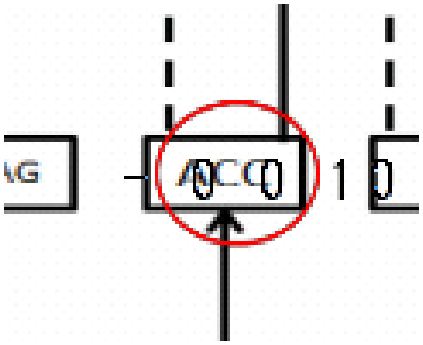
開始番地	機械語プログラム		アセンブリプログラム			コメント
00	64	80	LD	ACC,	[D1]	ACC <- [D1]
02	b4	81	ADD	ACC,	[D2]	ACC <- ACC + [D2]
04	74	82	ST	ACC,	[ANS]	[ANS] <- ACC
06	0f		HLT			停止



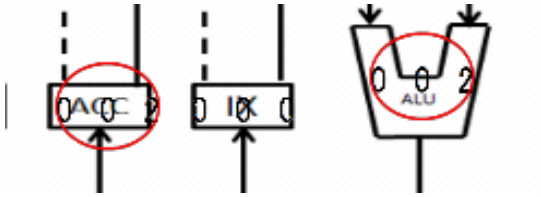
(a) 数値の入力



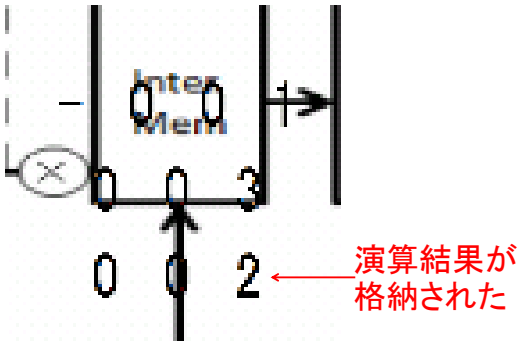
(b) 数値の反映(拡大表示)



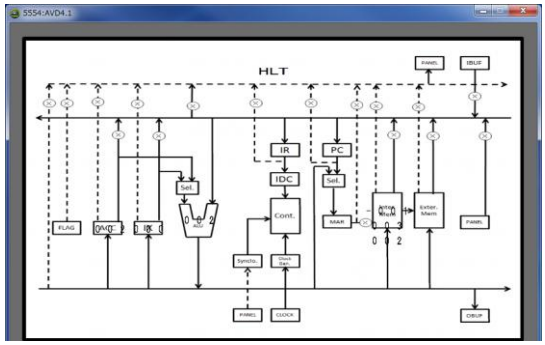
(c) LD命令(拡大表示)



(d) ADD命令(拡大表示)



(e) ST命令(拡大表示)



(f) HLT命令

図 14：2 値の加算プログラム - 実行

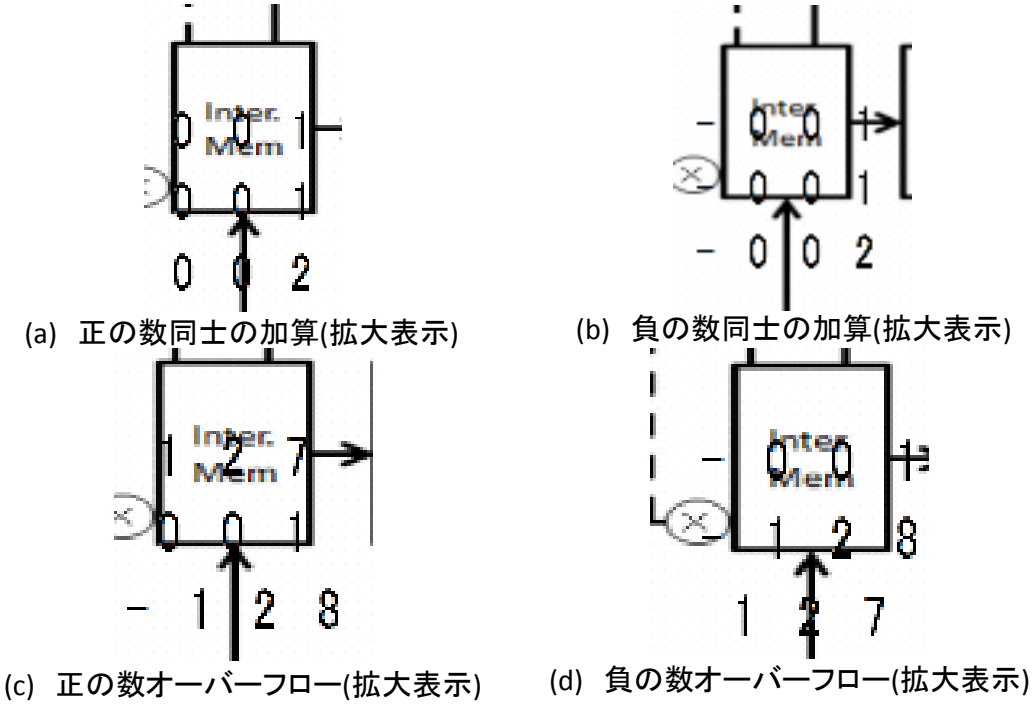


図 15：他の数値での実行結果

5.1.2 N 個の中の最大値

40 番地にデータの数，80 番地から順に比較する数字を入力し，c0 番地に最大値を格納する．N と，80 番地以降の数値は任意で変更することができる．プログラムとしては表 5 のように示すことができる．アセンブリプログラム欄の[N]は 40 番地，[IX+DATA]は 80 番地以降，[MAX]は c0 番地を指す．実行過程と結果を図 16 と図 17 に示す．

表 5：N 個の中の最大値のプログラム

開始番地	機械語プログラム		アセンブリプログラム				コメント
00	62	80		LD	ACC,	−128	ACC <− −128
02	6c	40		LD	IX,	[N]	IX <− [N]
04	aa	01		SUB	IX,	01	IX <− IX − 1
06	f6	80	LOOP:	CMP	ACC,	[IX+DATA]	ACC と[IX+DATA]を比較
08	36	0c		BGE		SKIP	IF ≧ GO TO SKIP
0a	66	80		LD	ACC,	[IX+DATA]	ACC <− [IX+DATA]
0c	aa	01	SKIP:	SUB	IX,	01	IX <− IX − 1
0e	32	06		BZP		LOOP	IF ≧0 GO TO SKIP
10	74	c0		ST	ACC,	[MAX]	[MAX] <− ACC
12	0f			HLT			停止

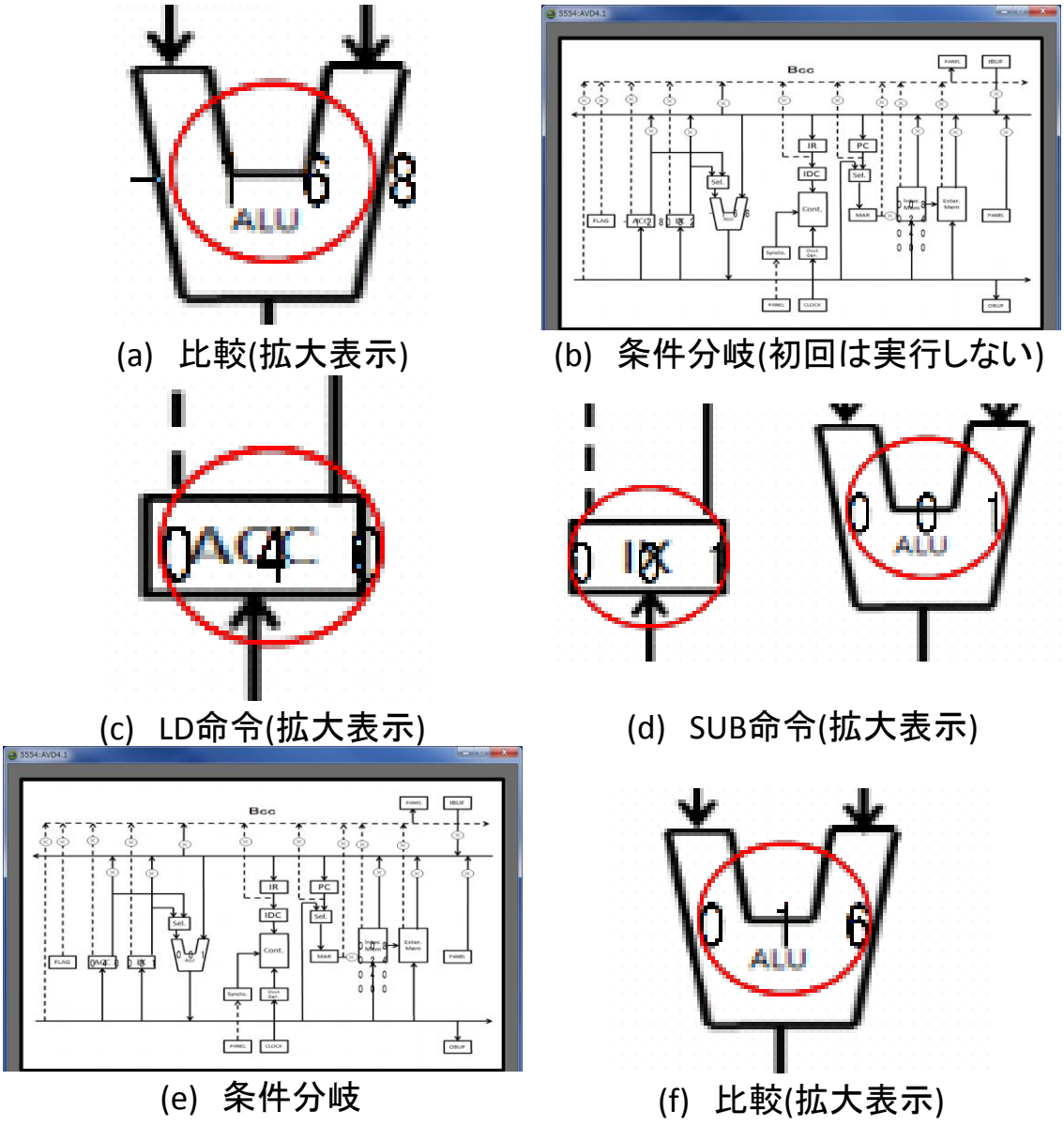


図 16：N 個の中の最大値 - 前半の処理過程

c0 番地に 03 を入力し，3 つの値で比較を行う．80 番地に 08，81 番地に 24，82 番地に 40 を格納した．この数値を使って最大値のプログラムを実行した．

表 5 の 00 番地の LD で ACC に-128 が格納される．02 番地の LD 命令では，40 番地に入力した 3 が IX に格納される．04 番地で，IX(3) - 1 が実行され，計算結果の 2 が IX に格納される．06 番地で比較の CMP 命令が行われる．CMP の ALU 上の動作は減算である．よって，最初は-128 - 40 が実行される．その結果が図 16 の(a)である．08 番地の分岐命令 BGE は，(b)に示すように，初回時は必ず実行されない． KUE-CHIP2 は先ほどの比較でオーバーフローが起こり，BGE の実行条件に当てはまらないからである．よって，次に 0a

番地の LD 命令が実行される．(c)のように，40 が ACC に格納される．そして，0c 番地の SUB 命令が行われる．(d)のように 2 - 1 の結果である 1 が IX に格納される．0e 番地の分岐条件である(e)は実行される．そのため，(f)のように 40 - 24 の演算を行う 06 番地の CMP 命令が実行される．このように条件分岐命令を実行しながら処理を続ける．後半の処理を図 17 に示す．

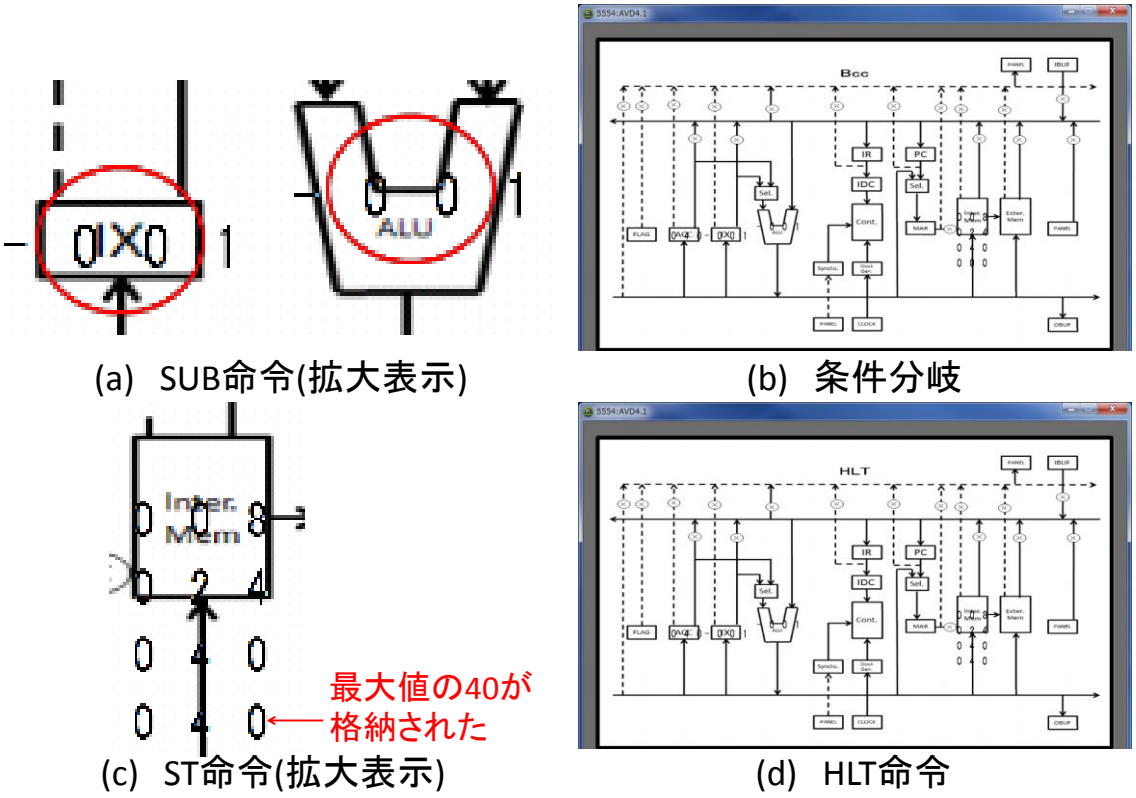


図 17：N 個の中の最大値 - 後半の処理過程

図 17 の(a)の SUB 命令は，表 5 の 0c 番地のものに相当する．この SUB 命令で，IX の値が-1 になったことがわかる．表 5 の 0e に相当する，(b)の分岐条件は 0 以上のとき，ジャンプするものなので，今回の演算結果では実行しない．そして，(c)の ST 命令で，最大値である 40 が c0 番地に送られたことが分かる．最後に，(d)の HLT 命令を経て，プログラムが終了した．

以上の結果から，条件分岐の命令を含む，N 個の中の最大値のプログラムは実行できた．また，演算過程と数値の変化も可視化することができた．

5.1.3 1 から N までの和

1 から N までの和を計算する．N は 80 番地に入力した値とし，N までの和を計算し，結果を 81 番地に格納する．プログラムとしては以下の表 6 のようになる．アセンブリプログ

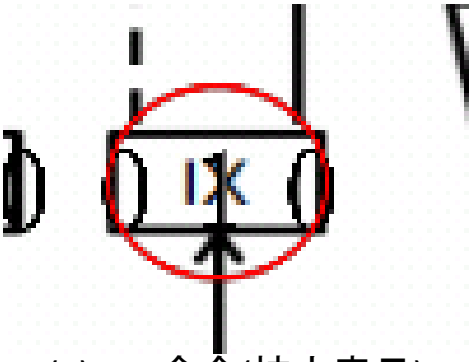
ラム欄の[N]は 80 番地，[SUM]は 81 番地を指す．また，命令を実行していく様子を，図 18 に示す．

最初の 00 番地の LD 命令を実行したものが(a)の画面である．IX に 10 が格納され，赤丸で強調されたことが分かる．次の 02 番地の LD 命令で ACC に 0 が格納される．その結果が(b)である．今回はデフォルト値で 0 が ACC に入っているので，強調表現はされなかった．(c)では 04 番地の ADD 命令が実行される．ACC(0) + IX(10)が計算され，ACC に結果が格納される．ACC に加算結果の 10 が格納されたことが強調されていることがわかる．(d)の SUB 命令では IX(10) - 1 が実行され結果が IX に格納される．計算結果の 09 が IX に格納されたことが強調されていることがわかる．(e)の BP 命令では，演算結果が 0 より大きいときに，指定した番地にジャンプする BP 命令である．先ほどの結果の 09 は 0 より大きいので，指定された 04 番地までジャンプする．(f)では，(c)と同様にして IX の値と ACC の値を加算し，ACC に格納する．(c)から(e)の処理を繰り返し，最後に(g)の HLT 命令までたどり着く．1 から 10 までの和である 55 が格納されたのがわかる．

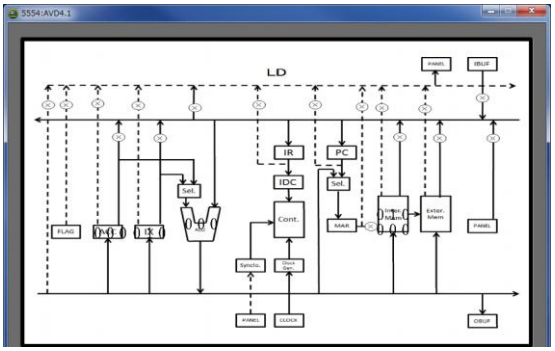
以上の結果から，シミュレーションと可視化できることを確認した．

表 6：1 から N までの和のプログラム

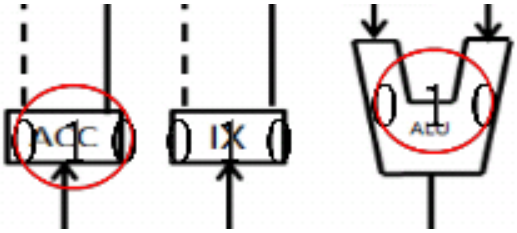
開始番地	機械語プログラム		アセンブリプログラム				コメント
00	6c	80		LD	IX,	[N]	IX <- [N]
02	62	00		LD	ACC,	00	ACC <- 0
04	b1		LOOP:	ADD	ACC,	IX	ACC <- ACC + IX
05	aa	01		SUB	IX,	01	IX <- IX - 1
07	33	04		BP		LOOP	IF > 0 GO TO LOOP
09	74	81		ST	ACC,	[SUM]	[SUM] <- ACC
0b	0f			HLT			停止



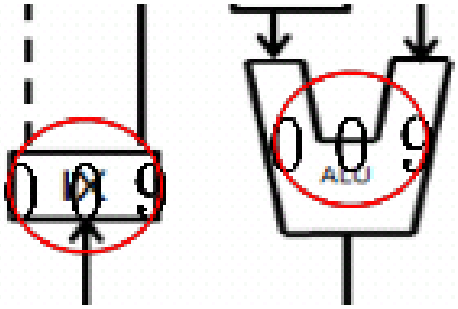
(a) LD命令(拡大表示)



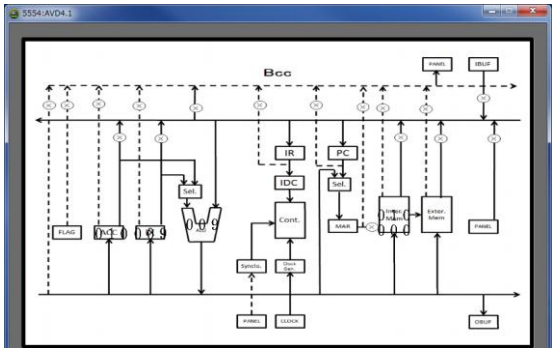
(b) LD命令



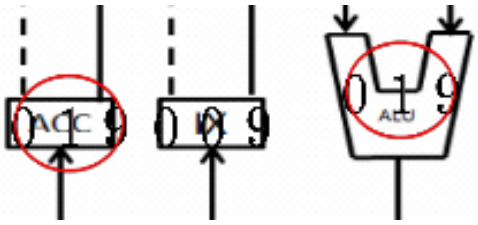
(c) ADD命令(拡大表示)



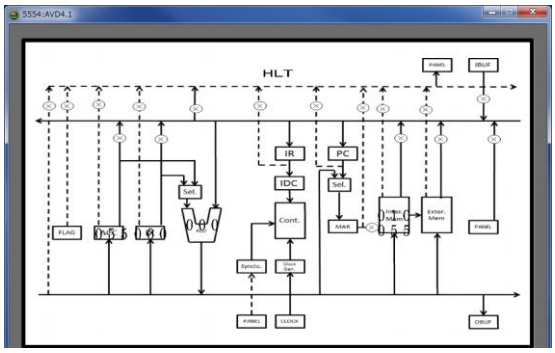
(d) SUB命令(拡大表示)



(e) 分岐条件



(f) ADD命令(拡大表示)



(g) HLT命令

図 18：1 から N までの和 - 実行結果

5.1.4　ユークリッドの互除法による最大公約数

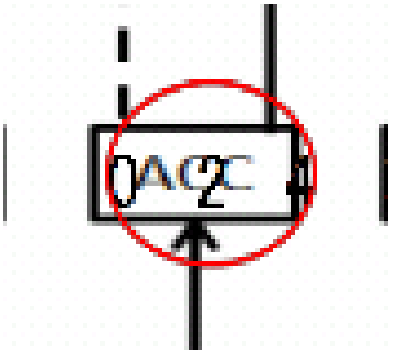
まずは，ユークリッドの互除法についての説明を行う．2 つの自然数 a，b (a ≥ b)があるとする．また，a ÷ b の余りを r とする．そして，a と b の最大公約数は，b と r の最大公約数に等しい．この性質から，b ÷ r の余り r'と，その余り r'を r で割ったときの余り r"を r'で割る…という計算を繰り返し，余りが 0 になったときの計算に使った除数が最大公約数であるというものである．

次に，具体的な数字で説明する．自然数 a，b のうち，a が 40，b が 24 とする．最初は 40 ÷ 24 = 1 … 16である．続けて，24 ÷ 16 = 1 … 8である．そして，16 ÷ 8 = 2 … 0である．このときの除数 8 が，今回の求めたい 40 と 24 の最大公約数である．

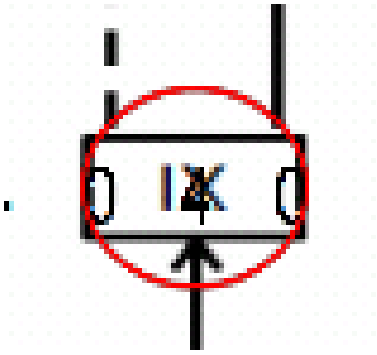
今回の例題では，80 番地と 81 番地に入力した値の最大公約数ユークリッドの互除法によって求め，82 番地に最大公約数を格納する． 80 番地と 81 番地の 2 つの値は任意で入力することができる．プログラムとしては，表 7 のように示すことができる．アセンブリプログラム欄の[A]は 80 番地，[B]は 81 番地，[GCD]は 82 番地を指す．今回は 24 と 40 を入力し，実行したときの過程及び，結果を図 19 に示す．

表 7：ユークリッドの互除法による最大公約数のプログラム

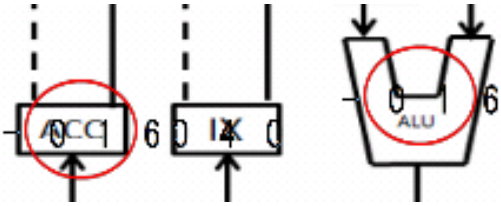
開始番地	機械語プログラム		アセンブリプログラム				コメント
00	64	80		LD	ACC,	[A]	
02	6c	81		LD	IX,	[B]	
04	a1		LOOP:	SUB	ACC,	IX	
05	32	04		BZP		LOOP	
07	b1			ADD	ACC,	IX	
08	c8			EOR	IX,	ACC	↑
09	c1			EOR	ACC,	IX	ACC と IX の値を交換
0a	c8			EOR	IX,	ACC	↓
0b	31	04		BNZ		LOOP	
0d	74	82		ST	ACC,	[GCD]	
0f	0f			HLT			



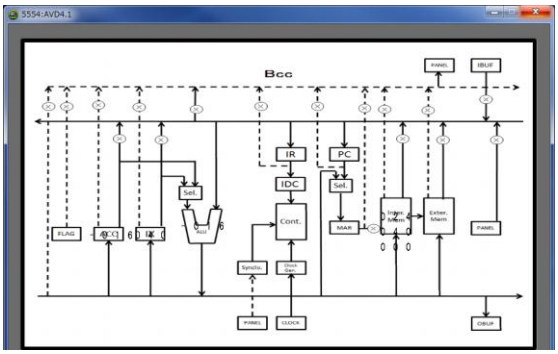
(a) ロード命令(拡大表示)



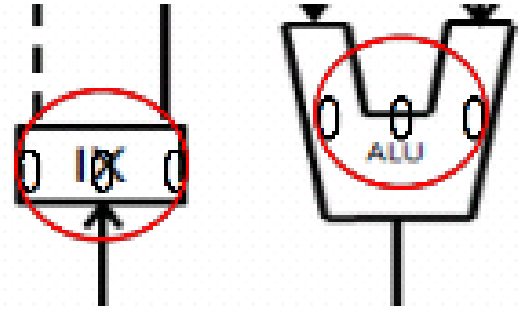
(b) ロード命令(拡大表示)



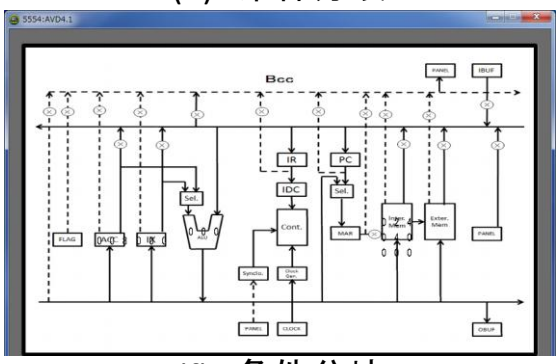
(c) SUB命令(拡大表示)



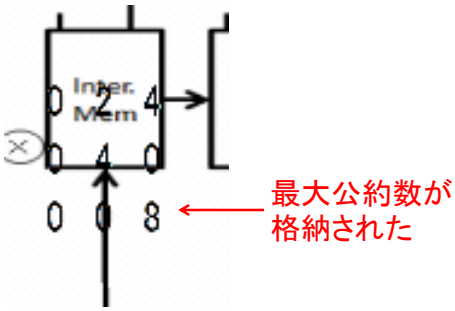
(d) 条件分岐



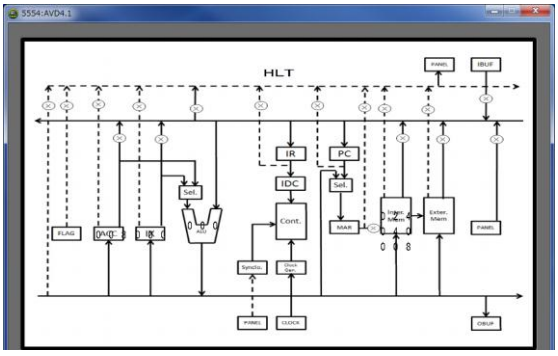
(e) EOR命令(拡大表示)



(f) 条件分岐



(g) ST命令(拡大表示)



(h) HLT命令

図 19：ユークリッド互除法 - 実行結果

表 7 の 00 番地の実行命令の様子が図 19 の(a)，02 番地に実行命令の様子が(b)である．共に，数値をレジスタにロードしてきている．04 番地である(c)では，24 - 40 の結果が負の数のため，05 番地の(d)条件分岐は行われない．07 番地の ADD 命令で-16 + 40 の結果 24 が ACC に格納される．次に，08 番地の IX(40) ^ ACC(24)の結果である 48 が IX に格納される．同様にして 09 番地の ACC(24) ^ IX(48)の結果である 40 が，ACC に格納される．また同様に 0a 番地の IX(48) ^ ACC(40)の結果である 24 が IX に格納される．このようにして，ACC と IX の値を入れ替える．このようにして，04 番地から 0b 番の命令を繰り返し，最大公約数を求める．最後の 08 番地の実行命令の様子を(e)に示す．この結果では剰余が 0 であることを示している．最大公約数を求める．(f)では，分岐条件に満たされないので実行されない．(g)では，解の 08 が 82 番地に送られている．(h)の HLT 命令でプログラムが終了している．以上から，実行過程と演算の様子を可視化することができた．

5.1.5 バブルソートによる整列

40 番地にデータの数，80 番地から順に比較する数字を入力し，昇順に整列した数値を，80 番地から順に格納しなおす．40 番地のデータの数と，80 番地以降の数値は任意で入力する．プログラムとしては表 8 のように示すことができる．アセンブリプログラム欄の[N]は 40 番地, [WORK1]は f0 番地, [WORK2]は f1 番地, [IX + DATA]は 80 番地以降を指す．実行過程と結果を図 20 に示す.

まず，40 番地にも 03 を入力し，ソートさせる数値を，80 番地に 07，81 番地に 06，82 番地に 05 を入力した. それが図 20 の(a)である. バブルソートによる処理が行われた場合，結果としては 80 番地に 05，81 番地に 06，82 番地に 07 となる．00 番地の LD 命令で IX に 40 番地の 3 が格納される．02 番地の SUB 命令で IX(3) - 即値(1)の結果の 2 が IX に格納される．04 番地の ST 命令で，ループ回数を一時的に保存するために，f0 番地に IX の値である 2 を格納する．それが(b)である．06 番地の EOR 命令で IX の値を 0 にしている．ここで ST IX, 00 としないのは，EOR IX, IX の方が，1 クロック動作が少なくなるからである．07 番地でキャリーフラグに 0 をセットし，08 番地で 80 番地の 07 をロードする．0a 番地で ACC(7)と 81 番地の 06 を比較する．ACC(7) - 6 = 1 なので，0c 番地の BLE は実行しない．0e 番地の ST 命令で ACC(7)を f1 番地に送る．その様子が(c)である．10 番地の LD 命令で 81 番地の 6 を ACC に格納する．12 番地の ST 命令で ACC の 6 を 80 番地に格納する．このときのメモリの 80 番地に 06，81 番地にも 6 が入っている状態である．14 番地の LD 命令で f1 番地の 7 を ACC に格納する．16 番地の ST 命令で ACC の 7 を 81 番地に送る．こうして，80 番地と 81 番地の値がソートされた．18 番地でキャリーフラグに 1 がセットされる．19 番地の ADD で IX(0) + 1 が演算され，結果の 1 が IX に格納される．1b 番地の CMP 命令で IX(1) - f0 番地(2)が実行される．1d 番地の BNZ は先の演算結果が 0 でないとき実行される. 先ほどの CMP 命令の結果は-1 だったので 08 番地にジャンプする．

先ほどの 08 番地から 16 番地の処理と同様にして，7 と 5 の比較を行い，値の交換を行う．そして，18 番地でキャリーフラグを 1 にし，19 番地の ADD 命令で IX(1) + 1 が実行され，IX に 2 が格納される．1b 番地の CMP 命令で IX(2) - f0 番地(2)が実行される．1d 番地の BNZ 命令は，先ほどの演算結果が 0 だったので実行されない．1f 番地の BNC もキャリーフラグが 18 番地の SCF 命令で 1 になっているため実行されない．21 番地の LD 命令で f0 番地の 01 が IX に格納される．23 番地の SUB 命令で IX(2) - 1 を実行し，1 を IX に格納する．25 番地の ST 命令で IX の値 01 が f0 番地に格納される．27 番地の BNZ 命令は，23 番値の SUB 命令の結果が 1 だったので，06 番地にジャンプする．同様にして，06 番地から 16 番地で，このときの 80 番地の 06 と 81 番地の 05 を比較し，値を交換する．そして，また同様に 25 番地まで実行する．27 番地の BNZ は，23 番地の SUB 命令で IX の値が 0 になっているので，実行されない．そして，29 の HLT 命令でプログラムが終了する．それが(d)である．メモリの 80 番地から 81 番地がソートされているのがわかる．以上から，バブルソートのシミュレートができ，かつ演算過程と結果を可視化することができた．

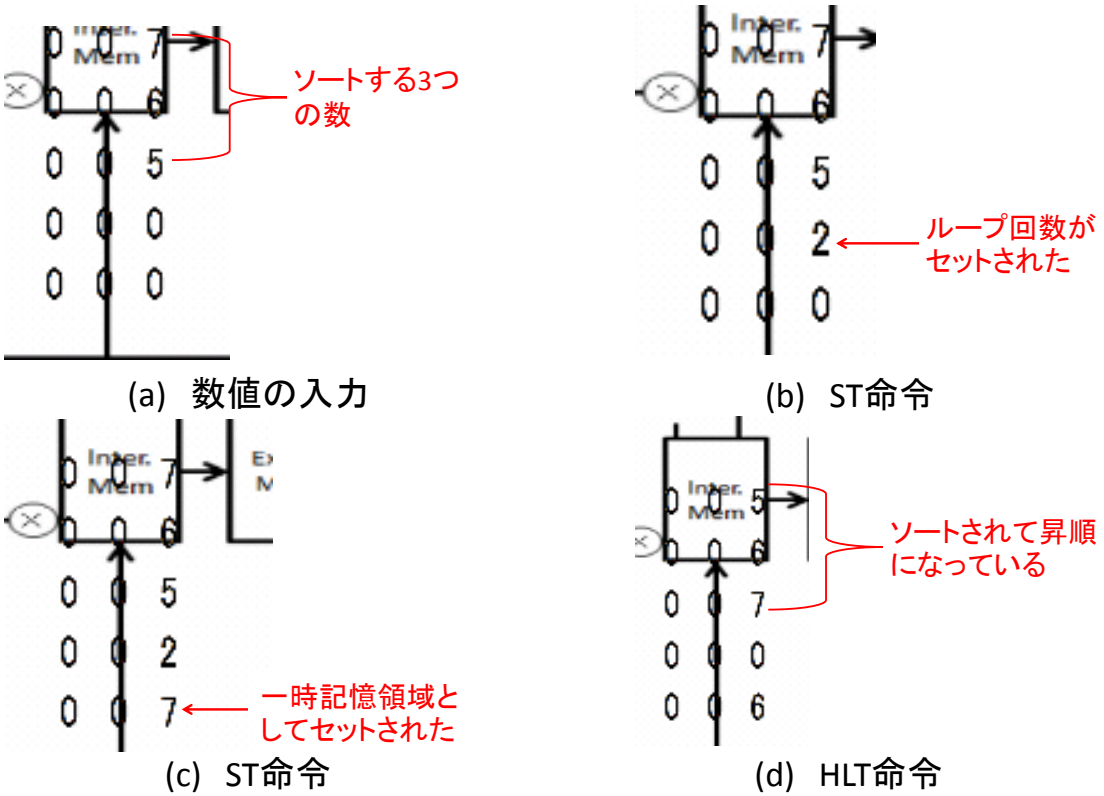


図 20：バブルソートによる整列 - 実行

表 8：バブルソートによる整列のプログラム

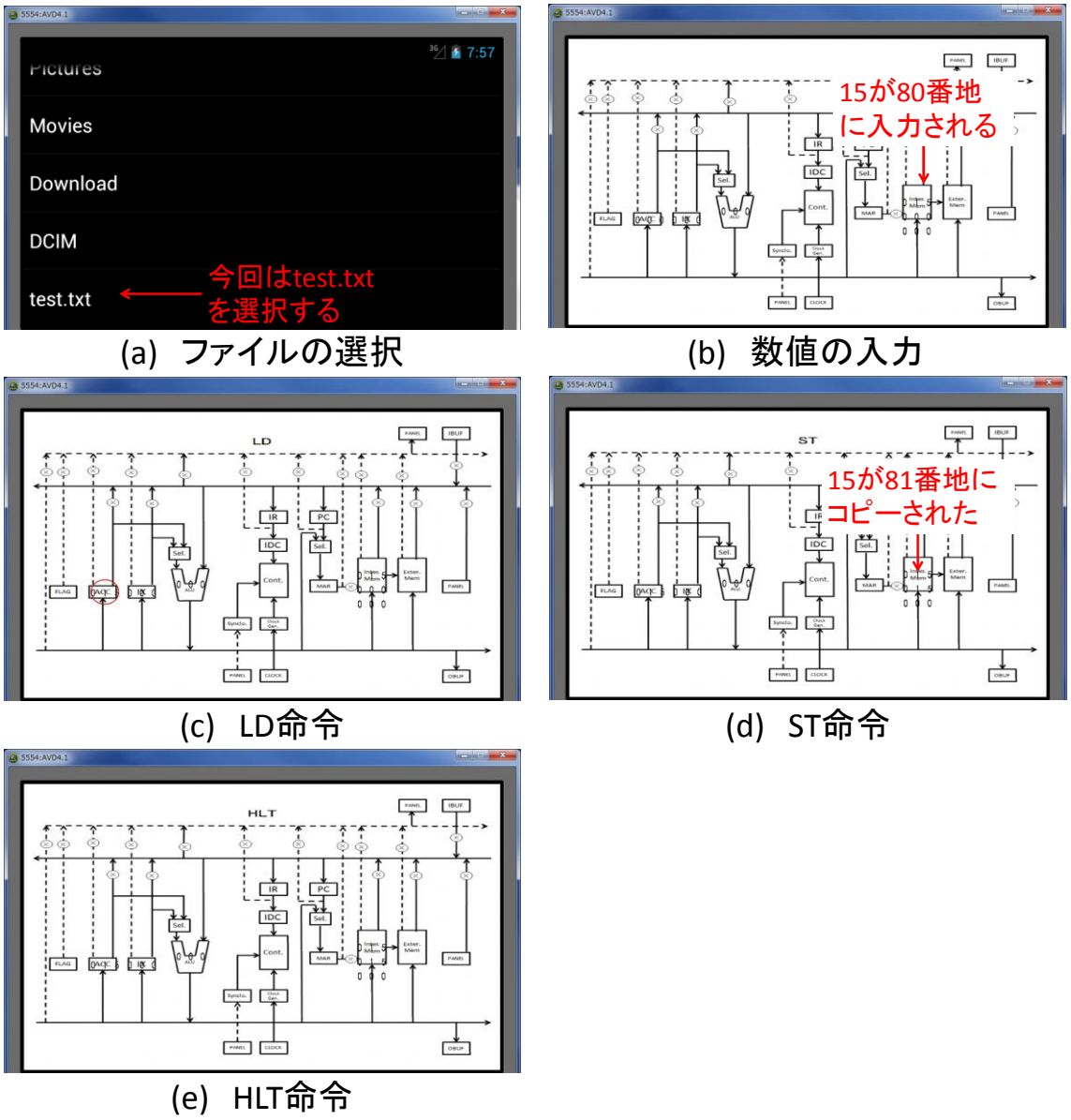
開始番地	機械語プログラム		アセンブリプログラム			
00	6c	40		LD	IX,	[N]
02	aa	01		SUB	IX,	01
04	7c	f0		ST	IX,	[WORK1]
06	c9		LP1:	EOR	IX,	IX
07	20			RCF		
08	66	80	LP2:	LD	ACC,	[IX+DATA]
0a	f6	81		CMP	ACC,	[IX+DATA+1]
0c	3f	19		BLE		SKIP
0e	74	f1		ST	ACC,	[WORK2]
10	66	81		LD	ACC,	[IX+DATA+1]
12	76	80		ST	ACC,	[IX+DATA]
14	64	f1		LD	ACC,	[WORK2]
16	76	81		ST	ACC,	[IX+DATA+1]
18	2f	f1		SCF		
19	ba	01	SKIP:	ADD	IX,	01
1b	fc	f0		CMP	IX,	[WORK1]
1d	31	08		BNZ		LP2
1f	35	29		BNC		FIN
21	6c	f0		LD	IX,	[WORK1]
23	aa	01		SUB	IX,	01
25	7c	f0		ST	IX,	[WORK1]
27	31	06		BNZ		LP1
29	0f			HLT		

5.2　ユーザー記述プログラム

5.1.1 から，5.1.5 までは，シミュレータ内にプリセットしておいたプログラムを実行し，表示していた．そこで，ユーザーの記述したプログラムを実行できるかどうかを試す．今回は 80 番地の数値を 81 番値にコピーするだけの簡単なもので検証した．表 9 に，そのプログラムを示す．アセンブリプログラム欄の[DATA]は 80 番地を指す．

表 9：80 番地の数値を 81 番地にコピーするプログラム

開始番地	機械語プログラム		アセンブリプログラム		
00	64	80	LD	ACC,	[DATA]
04	74	81	ST	ACC,	81
06	0f		HLT		



(a) ファイルの選択　(b) 数値の入力

(c) LD命令　(d) ST命令

(e) HLT命令

図 21：データのコピーを行うプログラム - 実行過程と結果

実際に，ソフトウェアにファイルを読み込ませ，実行させたときの過程と結果を図 21 に

示した．図 21 の(a)では，ファイルの選択画面である．今回は test.txt に，表 9 の内容を記述している．読み込ませ，実行した結果が(b)から(e)である．今回は 80 番地に 15 を入力した．その結果が(b)である．そして，順にロード命令，ストア命令を実行した．結果は(c)，(d)である．正しく動作していることが分かる．そして(e)のようにしてプログラムが終了する．この結果からプログラムの記述通り，シミュレートと，実行過程及び結果を可視化することができた．

5.3 考察

KUE-CHIP2 のシミュレータに 5 つの例題と，自作したプログラムを通して，各命令のシミュレートを行い，その演算過程で変化した数字を赤丸で強調し表現するとともに，演算結果を画面上に正しく表示することができた．

しかし，結果が正しく表示されたというだけであって，どの演算要素と通路を通っているのかが分からないため，各命令のデータパスの表示とは言えない．そのため，今の可視化表現とは違う工夫が必要だと考える．また，ソフトウェア自身にも，さらに工夫が必要であると感じた．

今後の課題として，データパスの可視化方法と，ソフトウェアの工夫を以下に重要度が高い順で提案する．

(1) 命令の詳細な表示
(2) 演算に必要な要素と通路に赤線を引く
(3) データパス上に数字が動くようにする
(4) メモリの画像をタッチすることで，メモリに任意の数値の入力を行えるようにする
(5) 制御信号の表示
(6) タブレット端末で実行する上での工夫

(1)は，シミュレートで画面上に表示しているのは命令の略記号のみなので，実行中の命令の詳細がよく分からないという問題がある．よって，より詳細に表示するようにソフトを改良する必要がある．(2)，(3)は，よりデータパス を理解してもらうための工夫である．これによって，演算に必要な機能モジュールと，データの移動経路を分かりやすくする．(4)は，今のソフトウェアの状態では，端末のメニューボタンを押す，出てきたボタンを押す，ポップアップウィンドウで入力する，というように，任意の数値を入力するまでのステップ数が 3 つあり，スムーズな作業とは言い難い．そこで，画面上のデータパスのメモリを タッチする，ポップアップウィンドウで入力する，というようにして 2 ステップで行えるようにする．また，入力方法も現在は 16 進数で固定しているが，10 進数でも数値を入力できるようにし，より直感的に操作できるようにしたい．(5)については，データパスではないものの，データパスと同様に重要なプロセッサを構成する要素である．データパスの可視化とともに，今後は制御信号の表示も視野に入れて開発を行っていきたい．(6)については，

現在のソフトウェアは Android OS で実行するアプリとして開発を進めているためである．実行環境としてはタブレットのような大きめの画面のものを想定しているが，機器ごとに画面の大きさが異なる．よってアスペクト比によっては，表示するデータパスが潰れて見難くなるということも十分ありえる．そこで，アスペクト比に応じて適切な表示を行えるようにする．また，タブレット端末ではなく，スマートフォンのような小型の機器でも実行する場合でも十分に見やすくする工夫が必要である．例えば，データパスの任意の個所を，あるいは，命令の演算に使われるデータパスを自動でピックアップし，その個所を拡大表示できるようにするなどである．

また，今回は 1 命令ごとに実行するシミュレータを作成したため，次に，1 クロックずつ実行するシミュレータを作成し，可視化できるようにする必要がある．

6. おわりに

本研究は，データパスを可視化することで，コンピュータの知識が少ない人にも，データパスの理解をしてもらうことを目標に進めてきた．

その上で，可視化プログラムの仕様と機能を設定した．まずは対象プロセッサを学習用ボードコンピュータである KUE-CHIP2 に設定した．また，可視化レベルを命令レベルと 1 クロックレベルの 2 つに定めた．また，プログラム中で用意した例題 5 つと，ユーザーが記述したプログラムのデータパスの可視化ができるようにするように取り決めた．そして，ユーザープログラムを実行できるようにするため，命令ごとにシミュレーションを行う KUE-CHP2 シミュレータと，そのシミュレータを通して，演算結果を画面上に表示する可視化プログラムを作成した．

プログラム中で用意した例題 5 つと，自作のプログラムを実行し，シミュレータが正しく動作し，また可視化も正しく動作することを確認した．しかし今の可視化方法では，目標に到達したとは言えず，また，1 クロックごとの可視化が実現できていない．

今後の課題として，1 クロックごとのシミュレータの作成と，より分かりやすい可視化方法の考案・工夫や，ソフトウェアをより使いやすくするということが挙げられる．

謝辞

本研究の機会を与えてくださり，貴重な助言，ご指導を頂きました山崎勝弘教授と孟助手に深く感謝いたします．また，事あるごとに相談に乗って頂き，貴重な助言を頂いた境氏，増田氏に深く感謝いたします．

最後に，高性能計算研究室の皆様に心より感謝いたします．

参考文献

[1] David A Patterson，John L Hennessy　著，成田光彰 訳：コンピュータの構成と設計(上)(下)第 3 版，日経 BP 社，2006.

[2] 山崎勝弘：電子情報デザイン実験Ⅲ　教育用ボードコンピュータ，立命館大学理工学部電子情報デザイン学科 実験テキスト，2010.

[3] 岩出秀平，清水徹：実用プロセッサ技術，ムイスリ出版，2009.

[4] 馬場敬信：コンピュータアーキテクチャ，オーム社，2000.

[5] 内田啓一郎，小柳滋：コンピュータアーキテクチャ，オーム社，2004.

[6] 北村俊明：コンピュータアーキテクチャの基礎，サイエンス社，2010.

[7] 堀桂太郎：図解 コンピュータアーキテクチャ 第 2 版，森北出版，2005.

[8] 柴田望洋：明解 Java 入門編，ソフトバンククリエイティブ，2007.

[9] 徳田雄洋：コンパイラの基礎，サイエンス社，2006.

[10] 安生真，柴田文彦，藤枝崇史：初歩からわかる Android 最新プログラミング，インプレスジャパン，2010.

[11] 中島安彦，横江宗太，株式会社パンカク：OpenGL で作る Android SDK ゲームプログラミング，インプレスジャパン，2011.