命令実行フェーズ

学籍番号 045713C:大城和也

実験実施日:平成 17 年 10 月 31 日 (月)
提出日:平成 17 年 11 月 7 日 (月)

実験共同者
045709E：上原　直久
045736B：知念　栄作
045739G：友寄雄一朗

1　目的

機械語 (マシン語) 命令をフェーズ毎に実行させ，その時のコンピュータ内部の状態を観測することにより，各フェーズでどのような処理が行われているかを調査し，機械語命令の実行の仕組みを理解することを目的とする．

2　概要

本実験では KUE-CHIP2 に格納された命令にかかるフェーズを確認する．そのためにまず KUE-CHIP2 に命令を格納し，SP ボタンを押してフェーズを 1 つずつ進める，その時には SEL スイッチを変えてそれぞれの状態を観測する．観測した値をそれぞれ見比べ，また KUE-CHIP2 リファレンス P73 の 7.6 実行命令フェーズを参考にコンピュータ内部での動きを理解する．最後にこれまでの仕上げとして割り算を行うアセンブラプログラムを考え実際に動かすということを行う．

3　実験結果

3.1　実験 (1) 以下に示すプログラムを用いて，減算命令 (SUB) の実行フェーズを観測せよ．ただし，ACC には，あらかじめ 07H を格納しておき，(7-5) の計算過程を観測するものとする．

番地	機械語	アセンブラ言語	
00	00	NOP	3 フェーズ
01	A2	SUB ACC, 05H	4 フェーズ
02	05		
03	0F	HLT	3 フェーズ

- 命令実行フェーズの観測手順

  (a) プログラムを KUE-CHIP2 に入力する．

  (b) REST スイッチを押し，全てのレジスタの内容を 0 にリセットする．

  (c) SEL スイッチを 0100 にし，ACC を選択する．

  (d) ACC に 07H を格納するために，DATA スイッチを 07H(00000111) にし，SET スイッチを押す．

(e) ボード上のフェーズ LED の P0 が点灯していることを確認した後，SP スイッチフェーズを 1 回押し，1 フェーズだけ実行させる．

- これにより，00H 番地の NOP 命令の P0(フェーズ 0) が実行され，ボード右上にあるフェーズ LED の P1 が点灯する．

(f) SP スイッチを 1 回押し，1 フェーズだけ実行させる．

- これにより，00H 番地の NOP 命令の P1(フェーズ 1) が実行され，ボード右上にあるフェース LED の P2 が点灯する．

(g) SP スイッチを 1 回押し，1 フェーズだけ実行させる．

- これにより，00H 番地の NOP 命令の P2(フェーズ 2) が実行され，ボード右上にあるフェーズ LED の P0 が点灯する．この時点で，NOP 命令の処理が終了したことになる．また，01H 番地の SUB 命令の P0(フェーズ 0) を実行する直前の状態になっている．

(h) SEL スイッチを操作し，SUB 命令の P0(フェーズ 0) を実行する直前の PC，FLAG，ACC，IX，DBi，DBo，MAR，IR の状態を観測する．

(i) 以後，SP スイッチを 1 回押し，1 フェーズだけ実行させるたびに上記の容量で各種レジスタ及びバスの状態を観測する．

- SUB 命令の P0(フェーズ 0) 実行後までの観測結果を表 3.1 に示す．空欄を埋めて表を完成させよ．

表 1: SUB 命令の実行フェーズ

フェーズ	LED	PC	FLAC	ACC	IX	DBi	DBo	MAR	IR
実行直前	P0 点灯	01	00	07	00	00	00	00	00
P0 実行後	P1 点灯	02	00	07	00	A2	A2	01	00
P1 実行後	P2 点灯	02	00	07	00	A2	A2	01	A2
P2 実行後	P3 点灯	03	00	07	00	05	02	02	A2
P3 実行後	P0 点灯	03	00	02	00	05	05	02	A2

3.2 実験 (2)4 つのプログラムについて，それぞれ 2 番目の命令の実行フェーズを観測し，表を完成させよ．また，それぞれのアセンブラプログラムに対する機械語プログラムを示せ．

それぞれの表を表 1~ 表 5 に示している．

3.2.1 LD 命令 (即値アドレス)

番地	機械語	アセンブラ言語
00	00	NOP
01	02	LD ACC, 05H
02	05	
03	0F	HLT

表 2: LD 命令 (即値アドレスモード) の実行フェーズ

フェーズ	LED	PC	FLAC	ACC	IX	DBi	DBo	MAR	IR
実行直前	P0 点灯	01	00	00	00	00	00	00	00
P0 実行後	P1 点灯	02	00	00	00	62	62	01	00
P1 実行後	P2 点灯	02	00	00	00	62	62	01	62
P2 実行後	P3 点灯	03	00	00	00	05	05	02	62
P3 実行後	P0 点灯	03	00	05	00	05	05	02	62

3.2.2 LD 命令 (絶対アドレスモード)

番地	機械語	アセンブラ言語
00	00	NOP
01	64	LD ACC, 07H
02	07	
03	0F	HLT

表 3: LD 命令 (絶対アドレスモード) の実行フェーズ

フェーズ	LED	PC	FLAC	ACC	IX	DBi	DBo	MAR	IR
実行直前	P0 点灯	01	00	00	00	00	00	00	00
P0 実行後	P1 点灯	02	00	00	00	64	64	01	00
P1 実行後	P2 点灯	02	00	00	00	64	64	01	64
P2 実行後	P3 点灯	03	00	00	00	07	07	02	64
P3 実行後	P4 点灯	03	00	00	00	FF	FF	07	64
P4 実行後	P0 点灯	03	00	FF	00	FF	FF	07	64

3.2.3 SCG 命令

番地	機械語	アセンブラ言語
00	00	NOP
01	2F	SCF
02	0F	HLT

表 4: SCF 命令の実行フェーズ

フェーズ	LED	PC	FLAC	ACC	IX	DBi	DBo	MAR	IR
実行直前	P0 点灯	01	00	00	00	00	00	00	00
P0 実行後	P1 点灯	02	00	00	00	2F	2F	01	00
P1 実行後	P2 点灯	02	00	00	00	2F	2F	01	2F
P2 実行後	P0 点灯	02	08	00	00	2F	2F	01	2F

3.2.4 AND 命令

番地	機械語	アセンブラ言語
00	00	NOP
01	E2	AND ACC, 05H
02	05	
03	0F	HLT

表 5: AND 命令の実行フェーズ

フェーズ	LED	PC	FLAC	ACC	IX	DBi	DBo	MAR	IR
実行直前	P0 点灯	01	00	00	00	00	00	00	00
P0 実行後	P1 点灯	02	00	00	00	E2	E2	01	00
P1 実行後	P2 点灯	02	00	00	00	E2	E2	01	E2
P2 実行後	P3 点灯	03	00	00	00	05	00	02	E2
P3 実行後	P0 点灯	03	01	00	00	05	05	02	E2

3.3 実験 (3) 下記のプログラムにおいて，IX=2 とした場合及び IX=1 とした場合のそれぞれについて，BZ 命令の実行フェーズを観測し，表を完成させよ．

それぞれの表を以下の表 6，表 7 に示す.

表 6: BZ 命令 (分岐不成立時) の実行フェーズ

フェーズ	LED	PC	FLAC	ACC	IX	DBi	DBo	MAR	IR
実行直前	P0 点灯	02	00	00	01	01	01	01	AA
P0 実行後	P1 点灯	03	00	00	01	39	39	02	AA
P1 実行後	P2 点灯	03	00	00	01	39	39	02	39
P2 実行後	P3 点灯	04	00	00	01	05	05	03	39
P3 実行後	P0 点灯	04	00	00	01	05	05	03	39

表 7: BZ 命令 (分岐条件成立時) の実行フェーズ

フェーズ	LED	PC	FLAC	ACC	IX	DBi	DBo	MAR	IR
実行直前	P0 点灯	02	01	00	00	01	01	01	AA
P0 実行後	P1 点灯	03	01	00	00	39	39	02	AA
P1 実行後	P2 点灯	03	01	00	00	39	39	02	39
P2 実行後	P3 点灯	04	01	00	00	05	05	03	39
P3 実行後	P0 点灯	05	01	00	00	05	05	03	39

3.4 実験 (4)8 ビットの 2 進数 m(データ領域の 0x00 番地に格納)，n(データ領域の 0x01 番地に格納) に対し，商 $m \div n$(小数点以下は不要) を求めるアセンブラプログラムを作成し，下記の場合の動作を確認しなさい．ただし，n=0 の時の商を 0xFF とすること．

(a) $m > n$ で，n が m を割り切れる場合の動作

(b) $m > n$ で，n が m を割り切れない場合の動作

(c) $m < n$ の場合の動作

(d) m = n の場合の動作

(e) n = 0 の場合の動作

また，このプログラムを解析し，C 言語によるプログラムに書き換えなさい．

番地	機械語	アセンブラ言語
00	C9	EOR IX, IX
01	65	LD ACC, 01H
02	01	
03	D0	OR ACC, ACC
04	39	BZ 0E
05	0E	
06	65	LD ACC, 00H
07	00	
08	BA	ADD IX, 01H
09	01	
0A	A5	SUB ACC,01H
0B	01	
0C	32	BZP 08H
0D	08	
0E	AA	SUB IX, 01H
0F	01	
10	0F	HLT

上記のプログラムが割り算を行うプログラムである．そのフローチャートが図 1 である．

表 8: 実行結果

	(a)	(b)	(c)	(d)	(e)
m(0x00)	0A	0A	0A	0A	0A
n(0x01)	02	03	0F	0A	00
IX	05	03	00	01	FF

これらを先の (a) から (e) までの値で出力したその結果が表 3.4 となる．その流れをフローチャートと共に説明する．

(a)∼**(c)** (a),(b),(c) は基本的に同じ動作をするのでまとめて説明する．(a)(b)(c) をすべてまとめるとなると m と n の条件としては m≥n ということになる．まず，IX を初期化し，ACC に n をロード，00 では無いため BZ をスルーし，次に ACC に m をロードする．ここでループに入る．IX はカウントとしてインクリメントし，ACC から n が引かれて ACC が

負の値になるまで続ける．この条件にして最後に IX から 1 を引くことでこれらの条件の時でもちゃんと値がでるようになっている．

(d) m と n の値を 0A とする．IX を初期化し，ACC に n をロード (ACC=0A)．条件分岐では ACC は 00 ではないため NO となり，ACC に m をロードする．次に IX に 1 を足し，ACC から n の値を引く．この時に ACC の値は 00 となるが次の条件分岐では 0 以上の時にはまた二つ前の命令に戻るようになっているのでもう一度処理を行うこのとき IX の値は 02．ACC は 00 から 0A の値を引かれることでマイナスのフラグを取り，先ほどの条件で NO となり次の処理に移り IX から 1 を引き IX の値は結局 00 をなって終了する．

(e) m の値を 0A，n の値を 00 とする．IX を初期化し，ACC に n の値をロードする，これにより ACC の値も 00 となる．次に条件分岐では ACC の値が 0 のため YES となり，IX から 1 を引く処理に飛ぶ．IX は初期化されて 0 だったためそこから 1 を引くことで FF の値となる．最後に HLT を通りプログラムは終了する．

```
-----------------------------------------------------------------
#include<stdio.h>

int main(){
  unsigned short IX, m, n; /* 変数宣言 */
  short ACC;

  m = 10;
  n = 02;
  IX = IX^IX; /* IX の初期化 */
  ACC = m;

  if(ACC == 0);
  else{
    ACC = n;
    do{
IX++; /* カウント */
ACC = ACC-n;
    }while(ACC >= 0);
  }
  printf("%x\n", --IX); /* IX から 1 を引いてその値を出力 */
}
-----------------------------------------------------------------
```

上記のプログラムはアセンブラプログラムを C 言語に置き換えたものである．KUE-CHIP2 における ACC,IX,0x00 番地，0x01 番地をそれぞれ変数に置き換え変数に置き換えている．m と n はすでに値が格納されているが計算するたびに違う値を代入した．BZ は if 文で BZP は do∼while 文で表している．これを動かした結果はアセンブラプログラム同様で表 8 と同じになる．

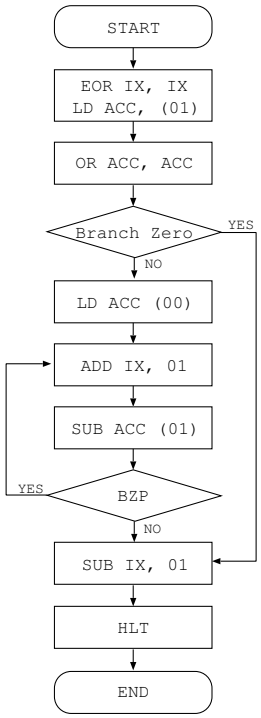

図 1: アセンブラフローチャート

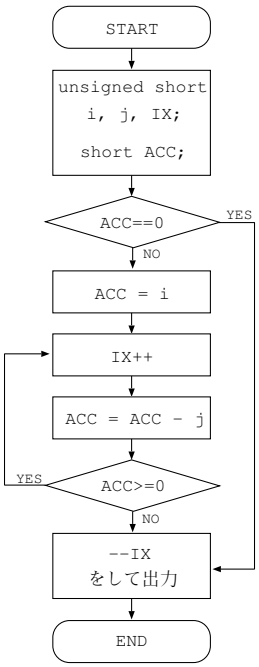

図 2: C 言語フローチャート

4 考察

4.1 実験 (1)，(2)，(3) について

4.1.1 SUB の実行フェーズ

(P0) P0 フェーズ実行前では命令 (NOP) がよばれたために PC の値が 1 つ増えてる．ACC が 07 となっているのは初めから 07 を格納したため．次に P0 フェーズを実行すると SUB 命令に移ったため PC がまた 1 つ増え 2 となり，MAR は命令のアドレスを保持するために 01 に，DBi と DBo はプログラムがよばれる際にそれぞれのバスを通ったため A2 となっていると考えられる．

(P1) P1 フェーズを実行すると変化するのは IR だけである．IR は命令を格納するので A2 という SUB 命令が格納されているのが分かる．

(P2) P2 フェーズを実行すると PC が 1 つ増え，DBi が 05 となり，DBo が 02 となって MAR が 02 となる．即値を扱う SUB は 2 語命令コードであるのでここでは次の行に移って 05 の即値を読み出しているのが分かる．DBi と DBo はそれぞれ 05 が読み出される際に DBo は ALU によって演算が行われた結果がそれぞれ通ったためだと考えられる．

(P3) P3 フェーズを実行するとここで結果を出力している．ALU で計算する際に 05 を扱うため DBi，DBo は 05 となっていると考えられる．

4.1.2 LD の実行フェーズ (即値アドレス)

(P0) P0 フェーズを実行すると PC は 1 増え，DBi と DBo には読み出される命令の 62 が格納されている．MAR は今の命令のアドレスを指しているのが分かる．

(P1) P1 フェーズを実行すると IR が 62 となって命令が格納されたのが分かる．

(P2) P2 フェーズを実行すると PC はインクリメントされ，DBi と DBo には次に呼び出される即値を格納している．MAR も PC から引き継いで 02 となっているのが分かる．

(P3) P3 フェーズを実行すると結果が ACC に格納されて LD の命令は終了する．

4.1.3 LD の実行フェーズ (絶対アドレス)

(P0) P0 フェーズを実行すると前回と同様にバスには呼び出される命令が格納され，PC がインクリメント，MAR には PC が指していたアドレスを保持.

(P1) P1 フェーズを実行すると IR に命令の格納

(P2) P2 フェーズを実行すると PC のインクリメント，MAR への保存，DBi，DBo には次に扱われるアドレスの数字が格納される．

(P3) P3 フェーズを実行すると MAR には先ほどの DBi，DBo での 07 というアドレスが格納され，そのアドレスの値が二つのバスを通っているので FF が表示されている．

(P4) P4 フェーズでは ACC に値を格納し，FF という値となる．

4.1.4 SFC の実行フェーズ

(P0) P0 フェーズを実行すると 2F という命令がバスを通ることが分かる．PC はインクリメントされ MAR には先の PC のアドレスを保存.

(P1) P1 フェーズを実行すると IR に命令が保存されていることが分かる．

(P2) P2 フェーズでは命令によって FLAC が 08 となっており，キャリーフラグのがたったことが分かる．

4.1.5 AND の実行フェーズ

(P0) P0 フェーズを実行することで PC はインクリメント，MAR には PC の先のアドレスが保持され，バスには読み出される命令が通っている．

(P1) P1 フェーズを実行すると IR に AND 命令の E2 が格納されている．

(P2) P2 フェーズを実行すると MAR に PC の保存，PC のインクリメント，DBi には ALU の演算のために読み出された値が DBo には ALU の演算での結果がそれぞれ格納されていると思われる．

(P3) P3 フェーズでは ACC に先ほどの結果を格納し，値が 00 であったため FLAC が 0 フラグを立てる．

4.1.6 BZ の実行フェーズ (分岐不成立時)

(P0) P0 実行前，それぞれが表のような値となっているのは先に SUB の命令を実行したためである．初めに IX を 2 としているので SUB の命令によって 1 引かれていることが分かる．フェーズを実行すると PC がインクリメントされ，MAR には先ほどの PC のアドレスが保存．二つのバスには BZ 命令の語が格納される．

(P1) P1 フェーズを実行すると IR には 39 という BZ の命令が格納されている．

(P2) P2 フェーズを実行すると PC を MAR に保持して PC をインクリメント，バスにはメモリから呼び出された 05 という値を格納している．

(P3) P3 フェーズを実行すると値はそのままで終了となる．フラグが発生しなかったためそのまま進むプログラムとなる．

4.1.7 BZ の実行フェーズ (分岐成立時)

(P0) この時は IX は 01 が格納されており，BZ 命令の直前では SUB によって 0 となっている．SUB となった時点で FLAC は 01 を指している．P0 フェーズを実行すると FLAC が 01 という以外は先の分岐不成立時と変わらない．

(P1) P1 フェーズの実行，FLAC が 01 ということ以外，不成立時と変わりなし．

(P2) P2 フェーズの実行，FLAC が 01 ということ以外，不成立時と変わりなし．

(P3) P3 フェーズの実行，FLAC が立っているため PC には DBi，DBo を通ってきた 05 というアドレスが PC に格納されていることが分かる．

4.1.8 フェーズの妥当性について

KUE-CHIP2 では，1 命令を P0，P1，P2，P3，P4 の最大 5 つのフェーズに分けているそのそれぞれの処理にも一定の規則性が見える．これらの規則性は後にパイプラインアーキテクチャにて説明するコンピュータの命令処理とほとんど同じだと考えられる．よってこのようなフェーズに分けることによりコンピュータの動きやパイプラインの考え方などが分かりやすくなるので KUE-CHIP2 でのこの分け方はよいと思われる．

4.2 C 言語とアセンブラについて

今回，できるだけアセンブラと同じような形にしたかったので C 言語とアセンブラ言語のプログラムのフローチャートはほとんど同じである．しかし，これは流れが同じなのであり，やはり違いはでてくる．Branch での処理はその前の演算された値しか判定できないが C の if 文などを用いればそのようなことは無い．さらに C 言語からコンパイルされアセンブラになったものは直接書いたものよりも遥かに長いアセンブラになっているであろう．

4.3 その他

今回，初めて DBi と DBo という言葉がでてきた，授業では触れられなかったため理解できたかは怪しいが自分なりの見解を．DBi と DBo はどちらもバスであり，それらはそれぞれの装置を結ぶ共通路である．そのため値を読み出したり，書き込む際には DBi と DBo は変化する．

5　調査課題

5.1　CPU の性能を表す指標

5.1.1　IPC について

IPC とは，1 クロック辺りに処理できる命令の数のことで，IPC は一般的に処理能力のパラメータとして扱われ値が大きいほど処理能力が高い．IPC を上げるためには，パイプラインやスーパスケーラなどのアーキテクチャでの工夫が必要となる．

IPC とは 1 クロック辺りの命令数であるので処理を早めるにはクロック周波数が大きく関わってくる．例えば IPC が 1 の CPU のコンピュータ A と IPC が 2 のコンピュータ B があってもコンピュータ A のクロック周波数が 2 倍以上となると IPC が 2 のコンピュータ B より先に処理を行えることもある．つまり，クロックと IPC は 2 つで 1 つの指標と考えた方がいいであろう．

5.1.2　クロック周波数について

クロックとは命令のタイミング合わせのための定期的な信号のことでこれが早ければ即ち命令の実行速度が早くなる．単位時間のクロック数のことをクロック周波数といい単位には Hz(ヘルツ) を用いる．周波数は値が大きい方が処理が早いといえ，IPC と同じく処理能力のパラメータとしては一般的である．

5.1.3　MIPS について

MIPS とは 1 秒間に実行できる命令数をメガ単位で表したものである．これと同様にギガ単位で表した GIPS などもある．

5.1.4　FLOPS について

FLOPS とは Floating point number Operations Per Second の略で 1 秒間に実行できる浮動小数点演算のことを表している．大型コンピュータに用いることが多い処理速度の指標である．

5.1.5　コマーシャルミックス

処理性能を調べる際にコンピュータの適用分野によって，よく使う命令と使わない命令がある．コマーシャルミックスは事務・会計など商業分野でよ

く使われる命令の組み合わせのプログラムの実行時間で処理性能を測るもので命令ミックスの 1 つである.

5.1.6 ギブソンミックス

コマーシャルミックスと同じく命令ミックスの 1 つで科学技術計算の性能を調べるものである.

5.2 パイプラインアーキテクチャ

パイプラインとは流れ作業である. コンピュータの命令処理には以下の 4 つの行程が上げられる.

- 命令フェッチ (主記憶装置からの命令の呼び出し F)
- 命令デコード (命令の解読, 処理対象のアドレス計算 D)
- 演算実行 (命令の演算の実行, 主記憶装置に対するデータの読み書き E)
- 結果の格納 (レジスタファイルに実行結果の格納. PC のセット W)

通常は 1 つの行程が完了するまで次の処理に移ることはできない. しかし, パイプラインは作業を幾つかの行程に分けてそれぞれを独立的に動かし, それぞれの空き時間を有効に使うことで処理の高速化を行う. これにより 1 つの命令を行っている際に次の命令の解釈を行ったりすることが可能となる.

以下の図 3 は上の 4 つの行程がそれぞれ同じ時間で終ったと考えた理想的なパイプラインとなっている. しかし, 実際の行程ではそれぞれの命令の長さが違ったり, 分岐命令などにより, 前後に依存関係が成り立つ場合などパイプラインの実行効率は理想よりは悪くなる.

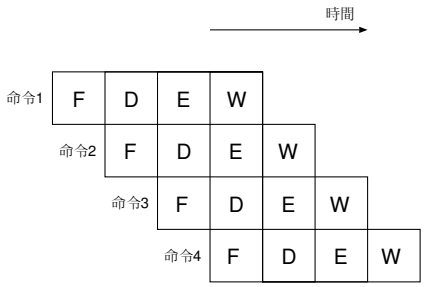

図 3: パイプライン

6　感想

今回はアセンブラのまとめでした．まとめということでプログラムを組んだりして中々面白かったです．今回のレポートでの一番の感想は Latex での表の配置はやっぱり難しいってことでした．時間がなくてレポートのレイアウトはかなり妥協気味に … 。まぁ、そこは勘弁ってところでお願いします．

参考文献

[1] 基本情報技術者試験サクセスガイドハードウェア, 一橋出版, 安藤明之

[2] 基本情報技術者試験合格教本, 技術評論社

[3] コンピュータアーキテクチャ, 電子情報通信学会, 著:坂井 修一

[4] http://e-words.jp/ “IT 用語辞典 e-Words”

[5] http://www.kecl.ntt.co.jp/car/parthe/html/lecture/95/kue2sfl.htm “Design of KUE-CHIP2”