情報工学実験２
命令実行フェーズ

035743A：比嘉雅樹

共同実験者： 035740F:根保光秀
実験日： 2004/11/15
提出日： 2004/11/22

1　実験の目的

機械語命令をフェーズ毎に実行させ、そのとき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調査し、機械語命令の実行の仕組みを理解する事を目的とする。

2　使用した器具

- KUE-CHIP2：PWB9408-050

3　実験結果

3.1　表１に示すプログラムにおいて、減算命令 (SUB) の実行フェーズを観測せよ。ただし、ACC には、予め 07H を格納しておき、計算過程も観測するものとする

表 1: プログラム

ARDS	DATA	OPECODE
00	00	NOP
01	A2	SUB ACC, 05H
02	05	
03	OF	HLT

減算命令 (SUB) の実行フェーズを観測した結果、下の表２に示すような結果になった。

表 2: SUB 命令の実行フェーズ表

フェーズ	LED	PC	FLAG	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 次に示す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せよ。

3.2.1 LD 命令（即値アドレスモード）の実行フェーズ

表 3: プログラム

ARDS	DATA	OPECODE
00	00	NOP
01	62	LD ACC, 05H
02	05	
03	OF	HLT

LD 命令（即値アドレスモード）の実行フェーズを観測した結果、下の表 4 に示すような結果になった。

表 4: LD 命令（即値アドレスモード）の実行フェーズ表

フェーズ	LED	PC	FLAG	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	02	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）の実行フェーズ

表 5: プログラム

ARDS	DATA	OPECODE
00	00	NOP
01	64	LD ACC, [07H]
02	07	
03	OF	HLT

LD 命令（絶対アドレスモード）の実行フェーズを観測した結果、次の表 6 に示すような結果になった。

表 6: LD 命令（絶対アドレスモード）の実行フェーズ表

フェーズ	LED	PC	FLAG	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 SCF 命令の実行フェーズ

表 7: プログラム

ARDS	DATA	OPECODE
00	00	NOP
01	2F	SCF
02	OF	HLT

SCF 命令の実行フェーズを観測した結果、下の表 8 に示すような結果になった。

表 8: SCF 命令の実行フェーズ表

フェーズ	LED	PC	FLAG	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 命令の実行フェーズ

表 9: プログラム

ARDS	DATA	OPECODE
00	00	NOP
01	E2	AND ACC, 07H
02	05	
03	OF	HLT

AND 命令の実行フェーズを観測した結果、下の表 10 に示すような結果になった。

表 10: AND 命令の実行フェーズ表

フェーズ	LED	PC	FLAG	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 下記のプログラムにおいて、IX=2 とした場合および 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、実行フェーズ表を完成させよ。

表 11: プログラム

ARDS	DATA	OPECODE
00	00	SUB IX, 01H
01	01	
02	39	BZ 05H
03	05	
04	0F	HLT
05	0F	HLT

BZ 命令の実行フェーズを観測した、次の表は IX=2 の時の実行フェーズ表である。

表 12: BZ 命令（IX=2）の実行フェーズ表

フェーズ	LED	PC	FLAG	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

次に、IX=1 の時の実行フェーズ表を示す。

表 13: BZ 命令（IX=1）の実行フェーズ表

フェーズ	LED	PC	FLAG	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 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。また、このプログラムを C 言語に書き換えなさい。ただし、n=0 のときの商を 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) n=0 の場合の動作

- アセンブラプログラム

表 14: 商を求めるプログラム

ARDS	DATA	OPECODE		
00	C9	EOR	IX,	IX
01	C0	EOR	ACC,	ACC
02	BD	LD	IX,	(01H)
03	01			
04	39	BZ	11H	
05	11			
06	6D	LD	IX,	(00H)
07	00			
08	AD	SUB	IX,	(01H)
09	01			
0A	3A	BN	10H	
0B	10			
0C	B2	ADD	ACC,	(01H)
0D	01			
0E	30	BA	08H	
0F	08			
10	0F	HLT		
11	62	LD	ACC,	(FFH)
12	FF			
13	0F	HLT		

- C プログラム

```
#include<stdio.h>

int main() {
  int ACC = 0;
  int IX = 0;
  int mn[2];

  printf("Input  m  --> ");
  scanf("%d", &mn[0]);
  printf("Input  n  --> ");
  scanf("%d", &mn[1]);
```

```
  IX = mn[1];
  if ( IX == 0 ) {
    printf("error\n");
    exit(1);
  }

  IX = mn[0];
  while( IX >= 0 ){
    IX -= mn[1];
    if ( IX < 0 ) {
      printf("\nResult --> %d\n",ACC);
    }
    else {
      ACC += 1;
    }
  }
}
```

- プログラムの説明
  今回作成したプログラムは、まず IX に n（0x00 番地）を読み込み 0 かどうか判別する。0 であった場合 ACC に 0xFF を格納する。0 でなければ IX に m（0x01 番地）を読み込み、m から n の値を引いていき、その結果が 0 より小さくならない限り、ACC に 1 づつ加算していく。0 より小さくなると終了する。

- C プログラムとアセンブラプログラムの違い

  - アセンブラでジャンプ命令をする際、プログラムが完成していないと、どこにジャンプすればよいか指定するのが難しいが、C ではプログラム途中でも容易にできる
  - 分岐命令をする際、アセンブラでは直前の結果との比較しかできないが、C では直前の結果でなくても比較ができる

4　報告事項

4.1　KUE-CHIP2 の P0〜P4 までの各命令実行フェーズについて、それぞれのフェーズにおいてどのような処理が行われているかを、実験結果から検討し、説明せよ。

- P0
  命令をフェッチするため、命令の入っているアドレス（PC に記憶されているアドレス）を MAR に送り、PC に 1 加算される。

- P1
  メモリを読み込み、MAR の内容が DBi を通して IR に転送され、次に実行される命令を解読可能な状態にする。なお、ここまではすべての命令において共通。

- P2
  - 1 語長命令
    命令が実行される
  - 2 語長以上の命令
    PC に記憶されているアドレスを MAR にセットし、読み込みまたは書き込みを行い、PC に 1 加算される。

- P3
  メモリの読み出しや演算処理を行う。結果は DBo を経由して MAR に送られる。

- P4
  演算結果を DBo を経由して ACC に格納する。また、演算結果にしたがって FR がセットされる。

4.2　前回の報告書で、パイプライン・アーキテクチャについて調査した。このパイプライン処理では、様々な原因により、処理の乱れを生じる危険性がある。このような、パイプライン処理における処理の乱れのことをパイプライン・ハザードまたは単にハザードという。パイプライン・ハザードの種類およびそれらの回避方法について調査せよ。

4.2.1 データハザード

- WAR ハザード（ライトアフターリードハザード）
  あるレジスタから値を読みこむ前に次の命令がそのレジスタを上書きする場合に起こる。

図 1: WAR ハザード

- WAW ハザード（ライトアフターライトハザード）
  あるレジスタへの最初の命令が書き込みを行った後に次の命令がさらに上書きするような場合に起こる。

図 2: WAW ハザード

- RAW ハザード（リードアフターライトハザード）
  あるレジスタに値を書きこんだ後に次の命令がレジスタを読む場合に、読み出し側の命令が書きこみが終わるまで待たずに読み込みを行う場合に起こる。

図 3: RAW ハザード

上記で挙げたハザードの内、RAW ハザード以外はレジスタリネーミング機能を使って回避できる。レジスタリネーミング機能とは、同時に実行する命令の数を増やすために、命令が参照するレジスタの名前を実行時に付け変える機能で、プロセッサには必ずついている機能。レジスタリネーミングを使

うと、スケジューラーは WAR および WAW ハザードが起きるときに、レジスタの対応づけを変更することで前後する命令の依存性を断ち切る事ができる。RAW ハザードは NOP を使ってレジスタの値が決定されるまで待つか、データフォワーディングという手法で解決できる。データフォワーディングとは、専用の回路を作り、演算結果等をすぐに次の命令で使えるようにするもの。図のように、データフォワーディング用の回路を作ることで EXE での結果をすぐに次の命令 ID で使えるようになる。

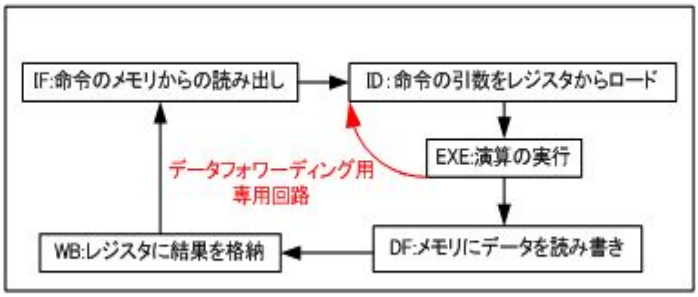

図 4: データフォワーディング

4.2.2 制御ハザード

ある命令を実行するか否かが、現在まだ実行中の他の命令の実行結果によって決まる場合に発生する。制御ハザードは、下記のような if 文で簡単に説明できる。

```
if(A)
{
   B
}
else {
   C
}
D
```

上の if 文では、A の結果が分からないと B が実行されるのか C が実行されるかがわからない。これと同じような事がパイプライン中でも起こり、パイプラインが止まってしまう。解決法としては次のような方法がある。

- 分岐予測

– 静的予測
分岐命令の場所などによって、どちらに分岐する可能性が高いかをプログラムレベル、コンパイルレベルで予測する。

– 動的予測
ハードウェアの予測回路によって、各分岐命令の挙動に応じて予測する。

分岐予測は、成功すればストールなしで動くが失敗すればストールが起こる。

- 遅延分岐
遅延分岐では、分岐の予測をせず、先に D（上記の if 文参照）を読み込む動作を行い、これにより分岐のストールを回避する。

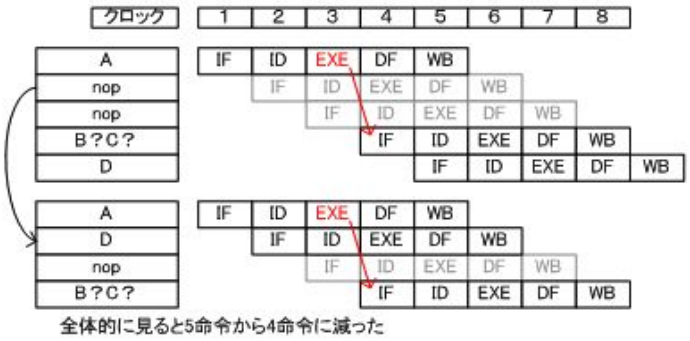

図 5: 遅延分岐

4.2.3 構造ハザード

パイプラインの２つ以上のステージが１つしかないハードウェア資源を取り合うために起こるハザード。解決法としては、十分はハードウェア資源を用意する事や、命令メモリとデータメモリをキャッシュレベルで分離する方法がある。

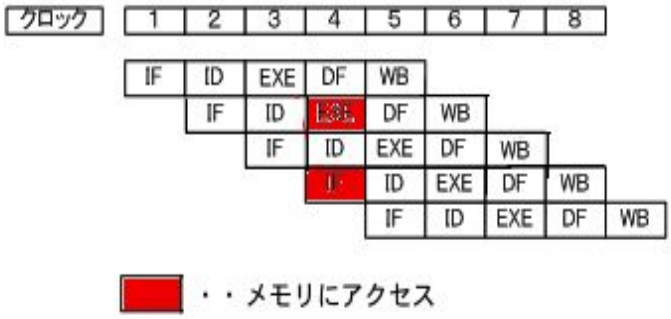

図 6: 構造ハザード

4.3 前回の報告書で IPC について調査した、IPC について以下の設問に答えよ

(a) IPC が 1 の CPU を載せたコンピュータ A と IPC が 2 の CPU を載せたコンピュータ B があり、両方のコンピュータで同じプログラムを同時に実行した。その結果、コンピュータ B の方が IPC が大きいにも関わらず、コンピュータ A の方が先に処理を終了した。この理由について考察せよ。

- CPU の処理速度は、IPC だけで決まるわけではない。動作周波数がいい例で動作周波数に極端に差があると、IPC が大きい方の CPU でも処理速度が遅くなる。

(b) 以下に示す 3 つのアーキテクチャは、いずれも IPC を向上させることによって、プロセッサの高速化を実現するアーキテクチャである。各アーキテクチャの特徴や違いを説明せよ。また、各アーキテクチャにおいて、IPC が向上する理由を説明せよ。

- スーパースカラ・アーキテクチャ
  複数のパイプラインを用意し、複数の命令を並列に処理する事ができる。しかし、分岐命令などの 2 つ同時に実行できない命令や、前の処理の結果で次の処理結果が変わってくる命令などは並列処理できない。

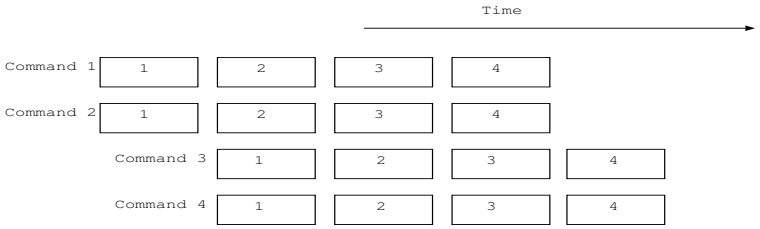

図 7: スーパースカラ・アーキテクチャ

通常のパイプライン処理より１クロックあたりの命令実行数が増えているので IPC が向上する。

- VLIW アーキテクチャ
  依存関係にない複数の命令を一つの命令としてまとめて同時に実行することで処理の高速化を図る。同時に実行される数は常に一定に保たれ、規定の数に達しない場合は「何もしない」命令で埋められる。１クロックで同時に複数の命令が処理できるため IPC は向上する。
- SMT アーキテクチャ
  複数のスレッドを、切り替えを行わずに１つのプロセッサ上で実行するアーキテクチャ。この手法では、トランジスタ数や消費電力を押さえながらパフォーマンスを向上させることができる。同時に複数のスレッドが実行されるので１クロックで処理される命令数が増えることになり、IPC が向上する。

(c) スーパーパイプラインアーキテクチャは、プロセッサの高速化を実現するアーキテクチャであるが、IPC は向上しない。その理由を説明せよ。
スーパーパイプラインアーキテクチャとは、CPU の処理を複数の細かいパイプラインステージに分割して行う事で高速化を図るアーキテクチャ。通常のパイプラインよりも処理の分割単位をさらに細かくして格段における処理内容を簡素化している。その分、格段の回路構成が簡略化されるので、さらにクロックを上げる事ができ、全体として処理能力の向上が期待できる。しかし、パイプラインの段数が多いだけに、命令の依存関係が発生しやすくなり、コンパイラによる最適化が非常に重要となる。また、分岐予測が外れた場合のペナルティも大きい。

スーパーパイプラインは、動作周波数は向上するが、１命令あたりに要するクロック数が増えるので IPC は向上しない。

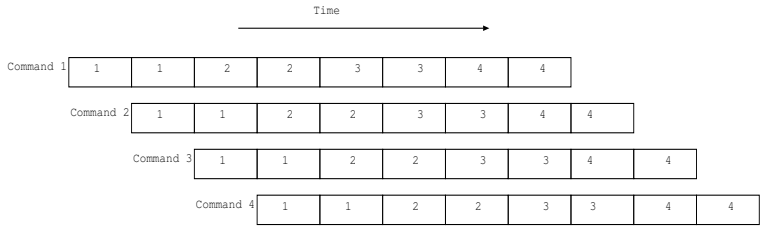

図 8: スーパーパイプラインアーキテクチャ

5　考察

今回の実験では、機械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で、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した。また、今回の実験でKUE-CHIP2 を使用した実験は終了するので、KUE-CHIP2 について調べてみた。

KUE-CHIP2(Kyoto University Education Chip 2) は、大学などでのコンピュー タ教育のための教材として開発された 8 ビットのマイクロプロセッサである。KUE-CHIP2 のアーキテクチャは非常に単純なもので 、その命令セットはコンピュータの基本構造と機能を学習するのに必要最 低限のものである。KUE-CHIP2 の特徴は、内部のレジスタの値を外部から観測、設定できることであり、バスの値を外部から観測することも可能である。
また、KUE-CHIP2 のメモリ空間は 512 バイトでバイト単位に指定される。0 番地から 255 番地はプログラム領域と呼ばれ、プログラムはこの範囲内に置かれなければならない。これは、PC が 8 ビットであるためである。256 番地から 511 番地まではデータ領域と呼ばれ、プログラムは格納できないが、データを格納する領域として利用できる。

参考文献

[1] サクセスガイド　ハドーウェア　著：安藤明之

[2] http://e-words.jp/　”IT 用語辞典”

[3] http://www.nifty.com/dictionary/　”@nifty デジタル用語辞典”

[4] http://yougo.ascii24.com/　”アスキーデジタル用語辞典”

[5] http://www.ddna.is.tsukuba.ac.jp/lecture/kue_chip/text.html#ch2
”マイクロコンピュータの動作”