情報工学実験2
命令実行フェーズ

035760A : 横田敏明

実験実施日：2004/11/12
レポート提出日：2004/11/19
共同実験：国吉貴文

1　実験概要

1.1　実験目的

機械語 (マシン語) 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．

1.2　プロセッサによる命令処理の流れ

プロセッサでの処理は，あらかじめ用意されたプログラムによって実行され，プログラム中の各命令は，基本的に 4 つのステージ (フェーズ) に分けて処理される．

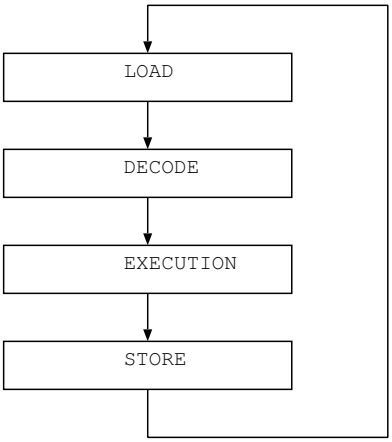

図 1: 命令処理の流れ

各命令に共通するステージは，命令の読み出し，命令の解読，命令の実行および実行結果の格納は，命令の種類によっては実行されない場合がある．

1.3　各ステージ

1.3.1　IF ステージ (Instruction Fetch)

IF ステージは，命令をメモリから読み出すステージである．読み出す命令の格納先アドレスは，プログラムカウンタに保持されている．

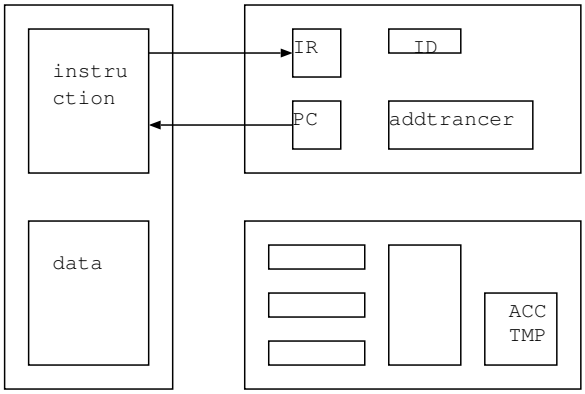

図 2: IF ステージの動作

1.3.2 ID ステージ

ID ステージでのプロセッサの動作を示す．ID ステージは，読み出した命令が，どのような命令であるかを解析するステージである．ID ステージでは，まず，命令レジスタ IR に格納された命令を，命令デコーダに渡し，命令の解析を行う．その結果に基づいて，以降のステージで実行される内容が決まる．演算の対象となるデータ (オペランド) は，即値として指定されている場合とレジスタやメモリに格納されている場合があるので，次にこの解析を行う．

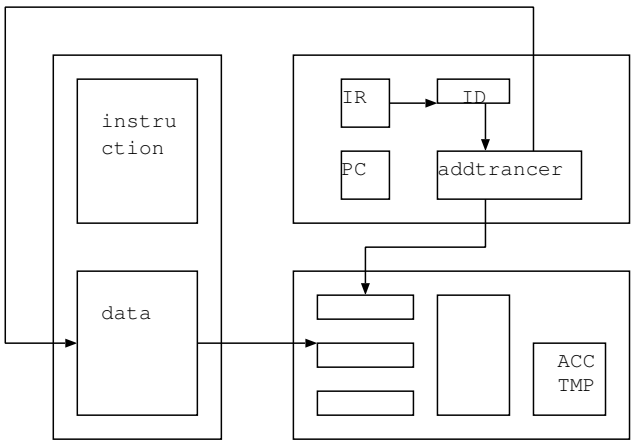

図 3: ID ステージ

1.3.3 EX ステージ

EX ステージは，命令で指定された処理を実行するステージである．EX ステージでは，命令の解析結果に基づき，演算回路を使用して必要な演算を実行する．演算には，四則演算，論理演算などの種類がある．演算の種類やデータの表現形式によって実行内容が異なるため，使用する演算回路は使い分けられる．

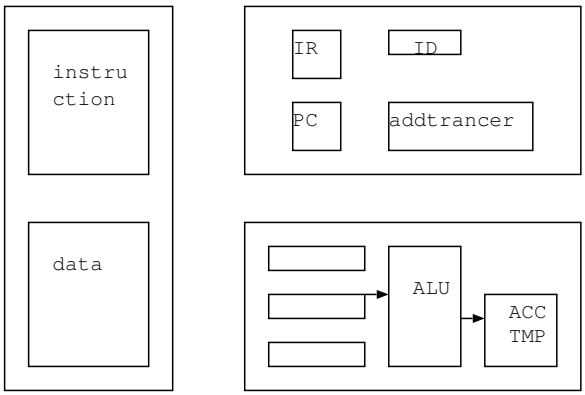

図 4: EX ステージ

1.3.4 ST ステージ

ST ステージは，命令の実行結果等をメモリやレジスタに格納するステージである．演算結果は，命令の指示に従い，プロセッサ内部のレジスタ，または指定されたメモリ上のアドレスに格納される．

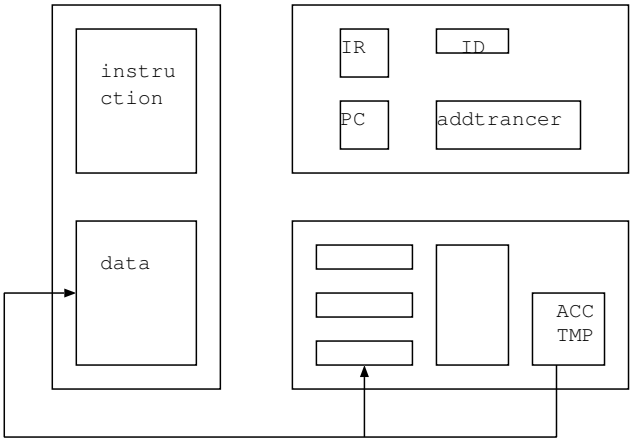

図 5: ST ステージ

2 報告事項

2.1 実験結果報告

(1) 以下に示すプログラムを用いて，減算命令の実行フェーズを観測せよ．

```
番地 機械語 アセンブラ言語
00   00     NOP
01   A2     SUB ACC, 05H
02   05
03   0F     HLT
```

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
P4 実行後	P0 点灯	03	00	02	00	05	05	02	A2

表 1: SUB 命令の実行フェーズ表

(2) 下記の 4 つのプログラムについて，それぞれ 2 番目の命令実行フェーズを観測し，実行フェーズ表を作成せよ．また，それぞれのアセンブラプロ

グラムに対応する機械語プログラムを示せ．(機械語プログラムはアセンブラと添えて記述した)

2.1.1 プログラム 1

```
00 00 NOP
01 62 LD ACC, 05H
02 05
03 0F HLT
```

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	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
P4 実行後	P0 点灯	03	00	05	00	05	05	02	62

表 2: LD(即値) 命令の実行フェーズ表

2.1.2 プログラム 2

```
00 00 NOP
01 64 LD ACC, [07H]
02 07
03 0F HLT
```

2.1.3 プログラム 3

```
00 00 NOP
01 2F SCF
02 0F HLT
```

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

表 3: LD(絶対) 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
P4 実行後	P0 点灯	02	08	00	00	2F	2F	01	2F

表 4: SCF 命令の実行フェーズ表

2.1.4 プログラム 4

```
00 00 NOP
01 E2 AND ACC, 05H
02 05
03 0F HLT
```

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
P4 実行後	P0 点灯	03	01	00	00	05	05	02	E2

表 5: AND 命令の実行フェーズ表

(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．なお，下記のプログラムにおいて，jp はラベルであり，BZ 命令

の分岐先のアドレスを表すものとする．機械語プログラム作製時に，各自で適当な値を jp に設定すること．

```
00 AA SUB IX, 01H
01 01
02 39 BZ 05H
03 05
04 0F HLT
05 0F HLT
```

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
P4 実行後	P0 点灯	04	00	00	01	05	05	03	39

表 6: BZ 命令 (不成立) 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
P4 実行後	P0 点灯	05	01	00	00	05	05	03	39

表 7: BZ 命令 (成立) の実行フェーズ表

(4)8bit 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/n(小数点以下不要) 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 のときの商を 0xFF とすること．

- m ¿ n で，n が m を割り切れる場合の動作:m=6,n=2 とすると IX=03 となった．
- m ¿ n で，n が m を割り切れない場合の動作:m=5,n=2 とすると IX=02 となった．

- m ¡ n の場合の動作:m=2,n=5 とすると，IX=00 となった.
- m = n の場合の動作:m=5,n=5 とすると，IX=01 となった.
- n = 0 の場合の動作:m=5,n=0 とすると，IX=FF となった.

```
ADRS DATA OPECODE
00   6D   LD IX, (01H)
01   01
02   BA   ADD IX, 00H
03   00
04   39   BZ
05   16
06   65   LD ACC (00H)
07   00
08   F1   CMP ACC - IX
09   3A   BN
0A   19
0B   6A   IX 00H
0C   00
0D   A5   SUB ACC -   (01H)
0E   BA   ADD IX 01H
0F   01
10   F5   CMP ACC - (01)
11   01
12   3A   BN
13   1B
14   30   BA
15   0E
16   6A   IX FF
17   FF
18   0F   HLT
19   6A   IX 00
1A   00
1B   0F   HLT
```

C 言語によるプログラム

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

```
int main()
{
  int ACC=0, IX=0, ADRS0x00,ADRS0x01;
  printf("input m >> ");
  scanf("%d",&ADRS0x00);
  printf("input n >> ");
  scanf("%d",&ADRS0x01);
  if(ADRS0x01 == 0){
    printf("Resume : FF");
    exit(0);
  }
  ACC = ADRS0x00;
  while(ACC >= ADRS0x01){
    ACC = ACC - ADRS0x01;
    IX = IX + 1;
  }
  printf("Resume : %d\n",IX);
}
```

```
[Toshiaki-YOKODAinZ32:~/exp2] j03060% a.out
input m >> 5
input n >> 2
Resume : 2
[Toshiaki-YOKODAinZ32:~/exp2] j03060% a.out
input m >> 6
input n >> 2
Resume : 3
[Toshiaki-YOKODAinZ32:~/exp2] j03060% a.out
input m >> 2
input n >> 5
Resume : 0
[Toshiaki-YOKODAinZ32:~/exp2] j03060% a.out
input m >> 5
input n >> 5
Resume : 1
[Toshiaki-YOKODAinZ32:~/exp2] j03060% a.out
input m >> 5
input n >> 0
```

```
Resume : FF
```

2.2 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．

フェーズ 0 では，フェッチ処理が行われている．LD(絶対) 命令を例にとってみる．実行直前では，プログラムカウンタの値は 01 を示し，DBi，DBo，は共に 00 が格納されている．フェーズ 1 に移行すると，プログラムカウンタは 1 つ進み，プログラムを読み出す．それによって，DBi,DBo に命令コードがインプットされる．
フェーズ 1 に移行すると，読み出されたプログラムを IR にインプットする．
フェーズ 2 に移行すると，プログラムカウンタが進み，アキュムレータに処理を施す準備を行う．つまり，命令を実行するために，DBi に読み出した値を代入し，DBo に結果を返し，ここでは読み出すアドレスを指定している．
フェーズ 3 に移行すると，指定されたアドレスから値を読み込む．読み込まれた値は一時的に保存され，アキュムレータにはまだ入っていない．
フェーズ 4 に移行すると，命令を実行し，アキュムレータに値を入れている．この結果から，ここで処理が行われるとわかる．

2.3 パイプライン・ハザード

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し，図表等を用いて分かり易く説明せよ．

パイプラインハザードの種類

2.3.1 構造ハザード:structural hazards

メモリへのアクセスの競合がおこることによっておこる．LD 命令の IF ステージは次のステージが終わるまでメモリにアクセスできない．BZ の OF ステージ，ADD の IF，OF ステージで起こる．

解決方法としては，命令フェッチとオペランドフェッチで用いるキャッシュを分離することで，命令フェッチとオペランドの競合をなくすことができる．これによって構造ハザードの問題を軽減できる．

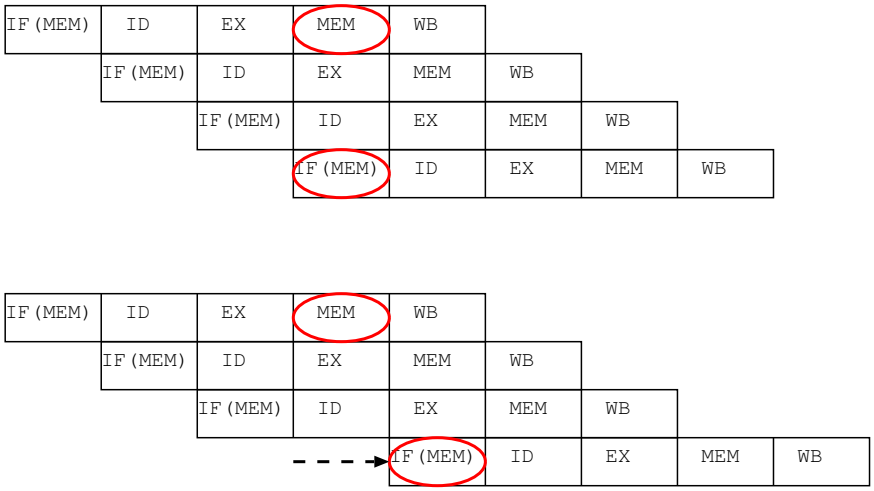

図 6: 構造ハザード

2.3.2 データハザード:data hazards

データ依存に起因して生じる．データのロードを行った後でなければ，そのデータを対象とする命令は実行できない．

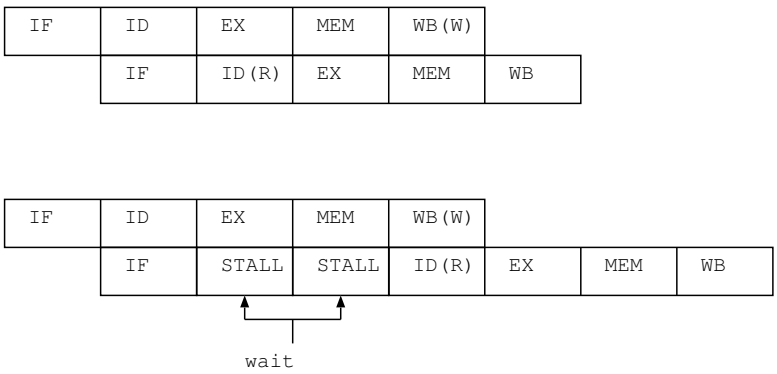

図 7: データハザード

解決法として，ALU チェイニング (データフォワーディング) がある．ALU の出力をレジスタなどに格納しないで直接同じ，もしくは異なる ALU の入力とする．

2.3.3 制御ハザード:control hazards

ある命令を実行するか否かが，現在まだ実行中の命令の実行結果に依存するときに起こる．

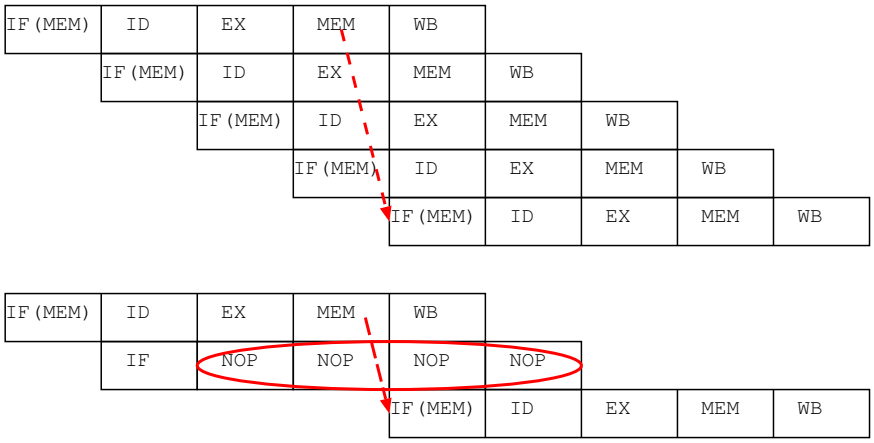

図 8: コントロールハザード

解決方法として，ループ制御，エラーチェック，プログラマの癖などで，分岐の傾向に偏りが生じる．過去の履歴から分岐方向を予測し，予測される方の命令をパイプラインに入れる．

3 IPC

3.1 問題 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 が大きいにもかかわらず，コンピュータ A の方が先に処理を終えた．この理由を考察せよ．

考えられる理由として，アーキテクチャの違いが挙げられる．まず，単純に A のコンピュータのプロセッサのクロック周波数が，B のプロセッサのクロック周波数を 2 倍以上上回っていた場合．クロック周波数と IPC の積によって実効速度が算出されるからである．

また，B の方が命令を実行する速さが上だとしても，1 つひとつの命令と処理をパイプラインで処理しなかった場合．かつ，A のコンピュータはパイプライン処理を行い，データ処理が 3 つ以上の部分に分割されていた場合，実効速度は B を上回る．

3.2 問題 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図表等を用いて分かり易く説明せよ．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:複数の実行ユニットを用いて演算を並列処理して高速化を図る仕組み。例として，PowerPC には、分岐、整数、浮動小数点の 3 個の実行ユニットが搭載されている。

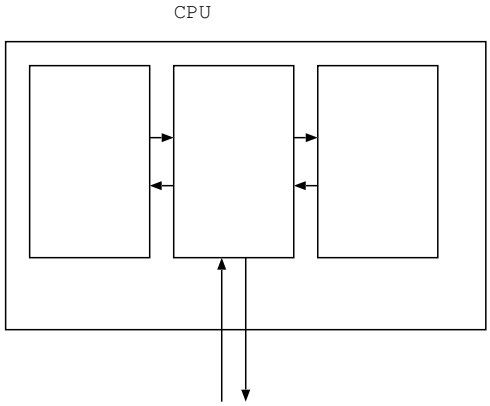

図 9: スーパースカラアーキテクチャ

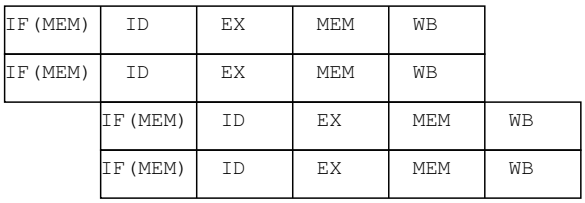

図 10: スーパースカラアーキテクチャの処理

- VLIW アーキテクチャ:演算器を複数持ったマイクロプロセッサのアーキテクチャ．同様に，スーパースカラも演算器レベルで並列処理を行う．VLIW アーキテクチャでは，並列実行のスケジューリングをコンパイラが行う．従って，スケジューリングをハードウェアで行うスーパースカラアーキテクチャより回路規模を小さくでき，動作周波数の上昇が見込める．問題点として，コンパイラの作成が複雑になり，オブジェクトコードに互換性がなくなるといった問題点がある．

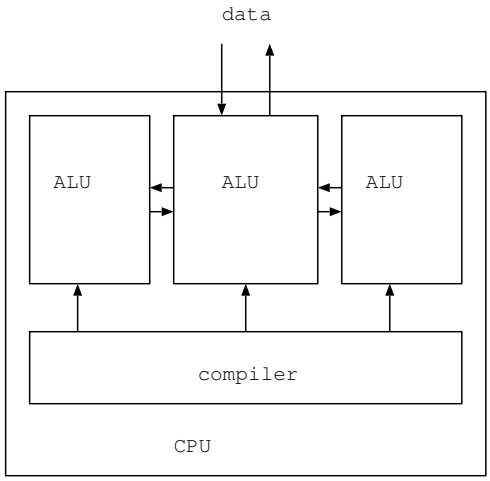

図 11: VLIW アーキテクチャ

- SMT アーキテクチャ:単一の CPU コアで複数スレッドを同時処理するアーキテクチャ．1 つの CPU を仮想的に複数の演算器をもつ CPU と見立て，分解された処理を並列して行う．最も回路を節約できるが，ハザードが起こりやすい．

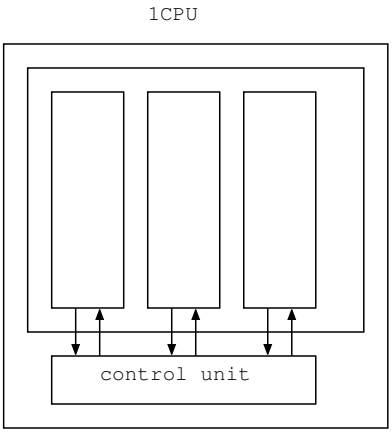

図 12: SMT アーキテクチャ

3.3 問題 C

以下に示すアーキテクチャは，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，図表等を用いて分かり易く

説明せよ．

- スーパーパイプライン・アーキテクチャ

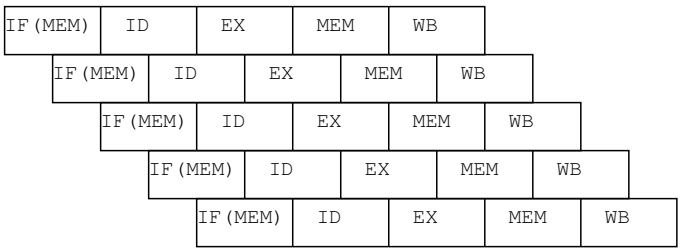

図 13: スーパーパイプラインアーキテクチャ

スーパーパイプラインアーキテクチャは，各命令サイクルを極限まで細分化し，パイプラインの段数を増やす方式である．IPC が向上しないのは，パイプラインの段数を増やしただけで，処理の内容は結局細かくした分だけ減る．ということが原因である．

4 考察

今回の実験で，フェーズごとの処理内容を細かく観察していき，コンピュータアーキテクチャの処理の段階ごとのメモリやデータの変化を見てきた．コンピュータの動作は，ある決められた種類の命令に分類でき，その命令は動作の順序が予めアーキテクチャの設計にあたって振り分けられている．これは動作の細分化を容易にすることができ，パイプライン処理をハードウェアのレベルで実現することが可能である．また，調査から，コンパイラを利用して，現在のアーキテクチャ，もしくは単一の CPU で並列処理を行うことが可能であることを知った．パイプライン処理は，作業を細分化して使用していない装置を最小限に抑え，処理の効率化を図る方法と理解した．

5 参考

- 現代計算機アーキテクチャ(ISBN-4-274-12962-4)
- http://cache.yahoofs.jp/seach/
- http://www.huis.hiroshima-u.ac.jp/ioss/ioss-1997/