[1]目的

マイクロコンピューターについて理解する。

[2]理論

現在、「コンピューター」と呼ばれているもののほとんどは、1940年代に Von Neumann が提唱したプログラム内蔵型コンピューターのモデルに準拠している。

Neumann が提唱したコンピューターとは、「内蔵されたプログラムに従って、命令を 1 つずつ順番に CPU（中央演算処理装置）に呼び出して実行するコンピューター」というものである。

今回実験で使用するコンピューターボードも、ザイログ社の「Z80」を CPU とするノイマン型のコンピューターである。

さて、使用されている Z80 の構造は、「演算器」「レジスタ」「入出力」「メモリバス」からなる CPU であり、レジスタ、メモリの内容により数々の演算処理を行うことができる。

演算処理はマシン語で行われる。また、データそれ自体には「プログラム」「データ」の区別は存在しない。

演算処理は「クロック」が入力されると実行される。クロックにパルス信号を任意に入力することにより、逐次実行（STEP）が可能になる。

■使用したボードのポート構成

入力ポート FF に 8bit の入力装置がついている。

出力ポート FF に 8bit の出力装置がついている。

[3]実験

■実験1

1.実験当日の日付を1バイトであらわし、入力ポートからAレジスタへ読み込む
2.Aレジスタの内容に、班番号を加算する
3.結果を出力する

以上のプログラムを作成せよ。

このプログラムをassembler言語で記述すると、以下のとおりになる。

```
LD      A,(0FFH)        日付を外部FFよりAに入力
OUT     (0FFH),A        Aの内容をFFに出力
ADD     A,06H           Aに6を足す
OUT     (0FFH),A        Aの内容をFFに出力
HALT                    プログラム中断
```

このプログラムをマシン語に翻訳すると、以下のとおりになる。

```
DB      FF              日付を外部FFよりAに入力
D3      FF              Aの内容をFFに出力
C6      06              Aに6を足す
D3      FF              Aの内容をFFに出力
76                      プログラム中断
```

このプログラムの実行結果は次ページのようになった。

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	MREQ		・		・		・				・		・		・				・		・		・		・		・		・				・		・	
2	IORQ								・								・													・	・					
3	RD		・				・		・		・				・				・				・		・				・				・			
4	WR																・													・	・					
5	M1	・	・							・	・							・	・					・	・							・	・			・
6	RFSH			・	・							・	・							・	・					・	・							・	・	
7	HALT																																			・
8	BUSAK																																			
9	D0																																			
10	D1																															・	・	・	・	・
11	D2																																			
12	D3																	・	・	・	・	・	・	・	・	・	・	・	・	・	・					
13	D4																																			
14	D5																																			
15	D6																																			
16	D7																	・	・	・	・	・	・	・	・	・	・	・	・	・	・	・	・	・	・	・
	アドレス	0	0	0	0	1	1	FF	FF	2	2	1	1	3	3	FF	FF	4	4	2	2	5	5	6	6	3	3	7	7	FF	FF	8	8	4	4	9
	データ	FF	DE	FF	FF	FF	FF	FF	77	FF	D3	FF	FF	FF	FF	FF	77	FF	C6	FF	FF	FF	6	FF	D3	FF	FF	FF	FF	FF	7D	FF	76	FF	FF	FF
	クロック回数						3		3						2		3	2					2						2		5	2		3		

■メモリへのデータの格納

1.プログラムの最後尾の次のアドレスを、データ格納用メモリアドレスとする

2.実験当日の日付を 1 バイトであらわし、それを入力ポートから A レジスタに読み込む

3.A レジスタの内容を、上で指定されたアドレスに「レジスタ間接」で格納する

以上のプログラムを作成せよ

このプログラムを assembler 言語で記述すると、以下のとおりになる。

```
LD      H,00H           レジスタ H に 00 を入力
LD      L,07            レジスタ L に 07 を入力
LD      A,(0FFH)        レジスタ A に外部 FF の内容を入力
LD      (HL),A          レジスタ HL の示しているメモリ番地にレ
                        ジスタ A の内容を入力
HALT                    プログラム中断
```

このプログラムをマシン語に翻訳すると、以下のとおりになる。

```
LD      H,00H           レジスタ H に 00 を入力
LD      L,07            レジスタ L に 07 を入力
LD      A,(0FFH)        レジスタ A に外部 FF の内容を入力
LD      (HL),A          レジスタ HL の示しているメモリ番地にレ
                        ジスタ A の内容を入力
HALT                    プログラム中断
```

このプログラムを実行してみたところ、メモリ 0007H にスイッチに入力した情報が入力されていた。

■相対ジャンプ

指導書に書かれたプログラムを実行し、動作を確認せよ。

指導書に書かれたプログラムは、以下のステップで構成されている。

1.レジスタ A に外部 FF の内容を入力

2.レジスタ A の内容を外部 FF に出力

3.1 に無条件ループ

このプログラムをマシン語に翻訳すると、下記のようになる。

DB	FF	レジスタ A に外部 FF の内容を入力
D3	FF	レジスタ A の内容を FF に出力
18	FA	PC に「FA」を足す
HALT		プログラム中断

■サブルーチン

実験時間が不足したため、指導書に載っているプログラムを翻訳するにとどまった。
以下に内容を示す。

実験4

(1)

3E 00	A に 00 を入力
D3 FF	A の内容を FF に出力
3C	A に 1 足す
C2 02 00	演算結果が 0 以外ならアドレス「0002」にジャンプ
76	HALT

(2)

3E 00	A に 00 を入力
D3 FF	A の内容を FF に出力
3C	A に 1 足す
CA 17 00	演算結果が 0 ならアドレス「0017」にジャンプ
06 00	レジスタ B に 00 を入力
0E 00	レジスタ C に 00 を入力
0D	C から 1 ひく
C2 0C 00	演算結果が 0 以外ならアドレス「000C」にジャンプ
05	B から 1 ひく
C2 0A 00	演算結果が 0 以外ならアドレス「000A」にジャンプ
C3 02 00	「0002」にジャンプ
76	HALT

[4]考察

■相対ジャンプにて、何故 FA を足すか

Z80 では、メモリの「どの番地を実行しているか」という情報を、プログラムカウンタ（PC）という特別なレジスタが管理している。

マシン語では、PC に対して足し算以外の概念は存在しない。

しかしながら、PC は1バイトの情報量しか持っていないため、FF に1を足すと 00 になってします。

このことを逆手に取り、プログラムカウンタに特定の値を足すことにより、次のニーモニックが格納されているところに持っていくことができる。

つまり、FA を足すことにより、00 番地に動作を移すことが出来るのである。

■現在のプロセッサの構造

現在、高速、高性能を歌ったプロセッサが数多く存在する。

しかしながら、基本は Z80 と同じ「マシン語逐次実行」である。

ではなぜ高速化できるのか？

私が調べた限り、「分岐予測」「ステージ数の増加」等の工夫が見られた。

本来、逐次に命令が実行されていくわけだが、ここに演算ユニットを多数配置したらどうなるであろうか？

命令は、1クロックで1命令…という早さでは実行できない。

何故なら、命令の解釈やメモリの読み込みで時間を食ってしまうからである。

なら、演算ユニットを増やし、見かけ上は1クロック1命令にしよう…というのが「ステージ数の増加」である。具体的には下図のように命令を重ねて実行していく。

UNIT1	1	1	1	4	4	4												
UNIT2		2	2	2	5	5	5											
UNIT3			3	3	3	6	6	6										
UNIT4																		
UNIT5																		
UNIT6																		
UNIT7																		
UNIT8																		
NONE	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6

UNIT1-8 ステージ数　／ NONE 単一演算ユニットしかない場合

1つの命令に4クロックかかるとすると…

ところが、この方式には欠点がある。仮に分岐があったとすると、そこで演算がとまってしまうのである。

UNIT1	1	1	1			4	4	4			
UNIT2		2	2	2			5	5	5		
UNIT3			☆	☆	☆			6	6	6	
UNIT4											
UNIT5											
UNIT6											
UNIT7											
UNIT8											

☆：分岐部分

なぜなら、分岐の条件が、現在演算中の要素により変わってくる可能性があるからである。

そこで、「分岐予測」という方式が提案された。
分岐予測とは、分岐されたことにして次の命令をやってしまう方式である。

UNIT1	1	1	1	4	4	4					
UNIT2		2	2	2	5	5	5				
UNIT3			☆	☆	☆	6	6	6			
UNIT4											
UNIT5											
UNIT6											
UNIT7											
UNIT8											

しかし、当然のように間違えた方向に進んでしまうことも考えられる。
すると、下図が示す通り、ペナルティーがかかってしまう。

UNIT1	1	1	1	4	4	4		4	4	4	
UNIT2		2	2	2	5	5	5		5	5	5
UNIT3			☆	☆	☆						
UNIT4											
UNIT5											
UNIT6											
UNIT7											
UNIT8											

☆の予測が間違いで、命令4と5がやりなおしな場合

このペナルティーの効率を良くすることにより、演算器は数十%もの演算時間を損していたりする。

そこで、米 intel 社は、同社の新しい CPU 設計思想「IA-64」にて、「プレディケーション」という概念を導入した。

プレディケーションとは、豊富なステージを利用し、条件分岐に遭遇した場合、両方とも実行してしまうという概念だ。

これを図に表すと、ステージこそ無駄になるが、CPU の応答速度は向上していることに気づく。

UNIT1	1	1	1	4	4	4	7	7	7		
UNIT2		2	2	2	5	5	5	8	8	8	
UNIT3			☆	☆	☆	6	6	6	9	9	9
UNIT4				4’	4’	4’					
UNIT5					5’	5’	5’				
UNIT6											
UNIT7											
UNIT8											

☆の予測が間違いで、命令4と5が正解、命令 4’ 5’ が不正解な場合

■並列処理に最適化されるマシン語

Z80 に限らず、現在のプロセッサは、ほとんどすべてがマシンコードが連続的である。

つまり、1命令/1つの要素で1クロックが終了しており、並列処理するには複雑なプロセスを CPU 側で行わなければならない。

そこで、intel の「IA-64」アーキティクチャでは、命令を4っつ入る大きな単位に分割し、並列に CPU に入力することにより、効率的に演算ユニットを駆使できる仕様が提案されている。

しかしながら、並列化されたマシン語を人間の手で作っていくのは不可能に近く、より優秀なコンパイラを開発していかなければならない。