情報工学実験2

ーコンピュータアーキテクチャ編ー
アセンブラプログラミング

075730G:澤岻千明

実験日 2008/10/23
締切日 2008/11/10
提出日 2008/11/10

共同実験者
075711A　山口　藍
075715C　嘉陽　裕香
075722F　斉藤由利絵

実験目的

アセンブラプログラムの作成、ハンドアセンブル (アセンブラプログラムを人手で機械語プログラムに直すこと)、実行の各作業を行うことにより、アセンブラプログラミングの流れを習得する。また、コンパイラとアセンブラの違いや高水準言語とアセンブリ言語の違いについて理解することを目的とする。

実験概要

今回の実験は前回にも用いた KUE-CHIP2 を使っての実験を行う。また、この他に D-A コンバータの回路を作成するためにブレッドボード、電気信号の波形観測のためにオシロスコープを用いる。ブレッドボードで D-A コンバータの回路を作成した後、プログラムを作成後、ハンドアセンブルし、KUE-CHIP2 へ入力する。プログラムを実行し、その結果をオシロスコープへ出力して波形を観測する。D-A コンバータとは、ディジタル信号をアナログ信号に変換する回路のことである。0 bit = 0 V、8 bit = 3.3V で出力される。bit が小さいほど 0 V、逆に大きい程 3.3V に近づくようになっている。これを利用し、オシロスコープで出力を波形にして観測するのである。

例としてのこぎり波を出力した後、各自でプログラムを考え矩形波、山形波、菱形波の波形を観測できることを確認する。

実験内容

(1) 図 1 に示した D-A コンバータをブレッドボード上に実現し、KUE-CHIP2 の出力ポートに接続せよ。また，リスト 2.1 のプログラムを KUE-CHIP2 に入力し、D-A コンバータのアナログ出力端子から図 2 に示したようなのこぎり波が出力されることを、オシロスコープを用いて確認せよ (報告の必要なし)。

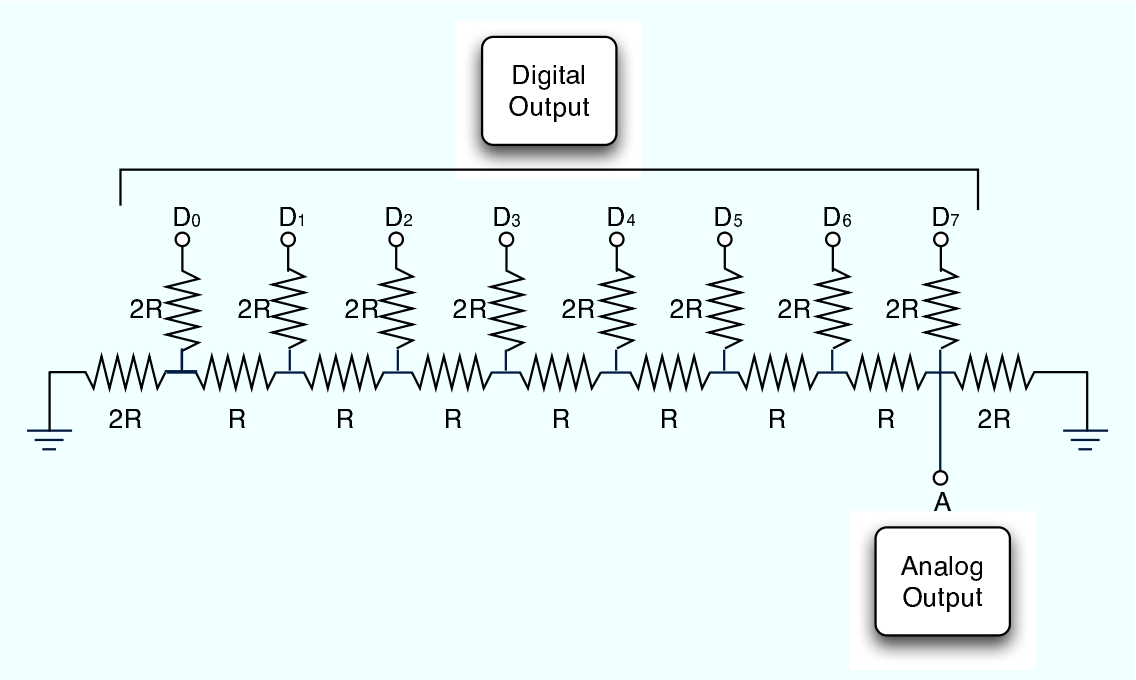

図 1: D-A コンバータ

リスト 2.1　のこぎり波を出力するプログラムの例

番地	機械語	アセンブラ言語		
00	C0		EOR ACC, ACC	＊ ACC のリセット (ACC ＝ 00H)
01	10	JP：	OUT	＊ ACC の値を 0BUF(出力ポート) に出力
02	B2		ADD ACC, 01H	＊ ACC の値をインクリメント (ACC ＝ ACC ＋ 01H)
03	01			
04	30		BA JP	＊ 01 番地に戻る
05	01			

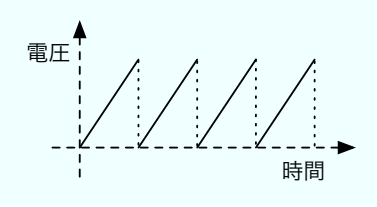

図 2: リスト 2.1 のプログラムの実行結果

(2) 図 3(a)〜(c) に示した各波形を出力するアセンブラプグラムを作成し、KUE-CHIP2 上で実行しなさい。なお、オシロスコープの設定および KUE-CHIP2 のクロック設定は、(1) ののこぎり波を表示させたときのままとし、変更しないこと

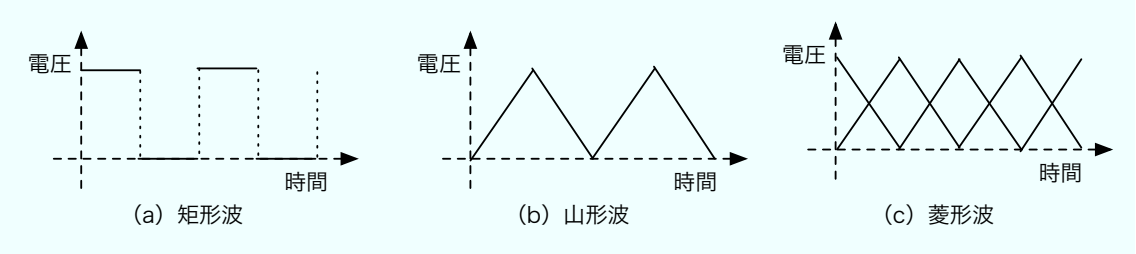

図 3: 実験用出力波形

実験結果

実施した実験項目 (実験 (1),(2)) を全て示し、以下の要領で各結果について報告せよ。

- 実験 (1) の結果について
  本実験の結果に関しては、報告を省略する。

- 実験 (2) の結果について

矩形波

番地	機械語	アセンブラ言語	
00	C9		EOR IX, IX
01	6A	JP3 :	LD IX, FFH
02	FF		
03	C0		EOR ACC, ACC
04	10	JP1 :	OUT
05	AA		SUB IX, 01H
06	01		
07	31		BNZ JP1
08	04		
09	6A		LD IX, FFH
0A	FF		
0B	62		LD ACC, FFH
0C	FF		
0D	10	JP2 :	OUT
0E	AA		SUB IX, 01H
0F	01		
10	31		BNZ JP2
11	0D		
12	30		BA JP3
13	01		

リスト 2.2:　矩形波を出力するプログラム

　矩形波のプログラムについてはリスト 2.2 に示す。
まずはインデックスレジスタ (IX) を初期化する。それから IX に値 FFH を書き込み、アキュムレータ (ACC) も初期化する。この時の ACC の値を出力する。この時点での値は 0 である。それから IX から 1 を引き、IX の値が 0 になるまで JP1 へ進む。IX が 0 になるまで引き、その間 ACC を出力し続けるので、その間 FFH が出力され続けることになる。IX が 0 になるとこの処理は終り、次の処理へと進む。

　ここで、また IX に値 FFH を書き込み、ACC にも FFH を書き込む。このときの ACC の値を出力し、さきほどと同じように IX から 1 を引いていき、0 になるまで ACC の値を出力し続ける。IX の値が 0 になるとこのループから抜け出し、最初と同じ処理を繰り返すプログラムとなっている。

山形波

番地	機械語	アセンブラ言語	
00	C0	JP3：	EOR ACC, ACC
01	10	JP1：	OUT
02	B2		ADD ACC, 01H
03	01		
04	31		BNZ JP1
05	01		
06	01		EOR ACC, ACC
07	01		LD ACC, FEH
08	01		
09	01	JP2：	OUT
0A	01		SUB ACC, 01H
0B	01		
0C	01		BNZ JP2
0D	01		
0E	01		BA JP3
0F	01		

リスト 2.3:　山形波を出力するプログラム

山形波のプログラムについてはリスト 2.3 に示す。

ACC を初期化し、その値を出力する。ACC に 1 を足したものを再び出力し、これが FFH からオーバーフローするまで繰り返す。

ループから抜けた後、再度 ACC を初期化し、値 FEH を書き込む。この状態を出力し、次は ACC から 1 ずつ引いていったものを再び出力する。これを 0 になるまで繰り返し、0 になった後は ACC を初期化し、最初の処理を繰り返すプログラムとなっている。

菱形波

番地	機械語	アセンブラ言語	
00	C0		EOR ACC, ACC
01	10	JP：	OUT
02	C2		EOR ACC, FFH
03	FF		
04	10		OUT
05	C2		EOR ACC, FFH
06	FF		
07	B2		ADD ACC, 01H
08	01		
09	31		BNZ JP
0A	01		
0B	30		BA JP
0C	01		

リスト 2.4:　菱形波を出力するプログラム

菱形波のプログラムについてはリスト 2.4 に示す。

ACC を初期化したあと、その値を出力する。その次はこの値を FFH で EOR で反転させ、出力させる。さらに EOR で反転させた後、ACC に 1 を足し、出力させ EOR

で反転しこの値を出力させる。この処理を0になるまで繰り返し、0になった後、最初の処理へ戻り繰り返すプログラムとなっている。

考察

上記の実験結果に基づいて、以下の点について詳しく述べよ。

- 実験 (2) の考察について
  - 今回の方法によって出力可能な波形と出力不可能な波形について考察せよ。
    OUT 命令で ACC の値が出力されるので、一度に出力できる値は常に一つだけなのである。よって縦線、つまり時間軸に垂直な線は引けない。
- その他の考察について
  - 本実験を通して得られた知見について詳しく説明せよ。
    オシロスコープを用いて、計算を波形にして視覚的にするというのがとても新奇性を感じた。どのような動きをしているのか、というのが理解しやすく面白く実験に集中しやすかった。

調査課題

下記の各項目について調査し、その結果を報告せよ。

(a) アセンブリ言語およびアセンブラについて、以下の設問に答えよ。

① アセンブリ言語を学習する目的はどのような点にあるか、自分の考えを論じよ。
CPU が直接理解できるのは機械語のみで、他の言語は人が理解しやすくするために生まれた言語である、ということをはっきり理解できたのはアセンブリ言語のおかげだと思う。また、アセンブリ言語は機械語と１対１で対応するので、細かく制御するのに適している。

② アプリケーション・ソフトウェアの開発には通常、C 言語などの高水準言語 (高級言語) が用いられるが、アセンブリ言語が使用される場面もある。アセンブリ言語が使用される理由について論じよ。
アセンブリ言語を使用すると、CPU の命令を直接記述できる。他の言語よりサイズを小さくでき、実行速度が比較的速くなる。

③ アセンブラプログラムを機械語プログラムに変換するソフトウェアをアセンブラと言う。また、アセンブラと似た働きをするソフトウェアとして、コンパイラとインタプリタが挙げられる。これら３種類のソフトウェアにおいて行われる処理の特徴や違いについて詳しく説明し、それぞれのソフトウェアで処理される代表的なプログラミング言語を挙げよ。

アセンブラ
機械語とほぼ 1 対 1 で対応するアセンブリ言語を文字列で書き並べ機械語に変換プログラム。
例) CAP-X、CASL

コンパイラ

コンパイル前のプログラム (ソースコード) はまとめて変換され、実行時にはコンパイル後のプログラム (オブジェクトコード) を直接実行するので、インタプリタに比べ実行速度が速い。

例) C、Java

インタプリタ

プログラム言語で書かれたソースコードを一行一行、その意味を逐次解釈しながら順次実行していく。

例) Perl、Ruby、

(b) 次回の実験で詳しく調べるが、１つの機械語命令は、いくつかのフェーズに分けて実行される。これはどのおうなプロセッサ (CPU) に対しても共通に言えることである。このことを利用して処理能力を向上させるアーキテクチャの１つにパイプライン・アーキテクチャがある。パイプライン・アーキテクチャとはどのようなアーキテクチャか調査し、図表を用いて分かりやすく説明せよ。

命令の実行時間を短くするための技術の一つ。

「命令の読み込み」と「命令の実行」を並列 (同時) に行うもので、これにより１命令あたりの (見掛けの) 処理時間を短くできる。命令は以下のような工程に分かれている。

＊ IF (Instruction Fetch)
　— 命令を命令キャッシュから読み出す
＊ ID (Instruction Decode/register read)
　— 制御信号を生成し、レジスタ・ファイルをレジスタ指定子で参照する
＊ EX (EXecution/address calculation)
　— 数値の計算やロード・ストアのデータやアドレス・分岐先の計算を行う
＊ MA (Memory Acess)
　— ロード (メモリの読み出し)・ストア (メモリへの書き込み) を行う
＊ WB (Write Back)
　— レジスタにデータを書き込む

各工程が同時にデータを運んでいて、各工程が次の工程をパイプのように接続されていることから、パイプラインという。

初期のプロセッサの設計では、プロセッサはステップのすべてを次の命令に移る前に実行していた。だが、これでは回路の大部分が各ステップで待機状態になってしまう。実行やそのほかの段階では命令でコードの回路は待機状態になる。(図 4 参照)

時間	1	2	3	4	5	6	7	8	9	10
命令 1	IF	ID	EX	MA	WB					
命令 2						IF	ED	EX	MA	WB

図 4: パイプライン未処理の例

パイプラインを用いることで、一定数の命令がプロセッサ内で同時に動作できるようにすることで性能を向上させる。複数の命令を同時に「動作状態」におくことができる。どの命令も完了するのに同じ時間がかかるのだが、CPU は結果として命令を大幅に早く片付けることができ、より高いクロック速度で動作することができる。(図 5 参照)

時間	1	2	3	4	5	6	7	8	9	10
命令 1	IF	ID	EX	MA	WB					
命令 2		IF	ID	EX	MA	WB				
命令 3			IF	ID	EX	MA	WB			
命令 4				IF	ID	EX	MA	WB		
命令 5					IF	ID	EX	MA	WB	
命令 6						IF	ID	EX	MA	WB

図 5: パイプライン処理の例

感想

アセンブリ言語を考える時点で非常に手こずって大変でした。

合ってはいても、ハンドアセンブルでミスを連発してしまい終わるのに時間がかかって残念です。オシロスコープで波形が見れたときはとても面白く、これなら楽しくできる！と思ったのも束の間。アセンブリで手間取り思考停止に近い状態にまで陥りました。

この時点では次回は今回のよりは手際良くできるだろうと思っていました。現実は甘くないものです。

参考文献・引用文献

- LaTex -コマンド一覧
  http://www1.kiy.jp/~yoka/LaTeX/latex.html
- LaTeX コマンドシート一覧
  http://www002.upp.so-net.ne.jp/latex/
- Wikipedia
  http://ja.wikipedia.org/wiki/
- 教えて！goo
  http://oshiete1.goo.ne.jp/qa3453461.html