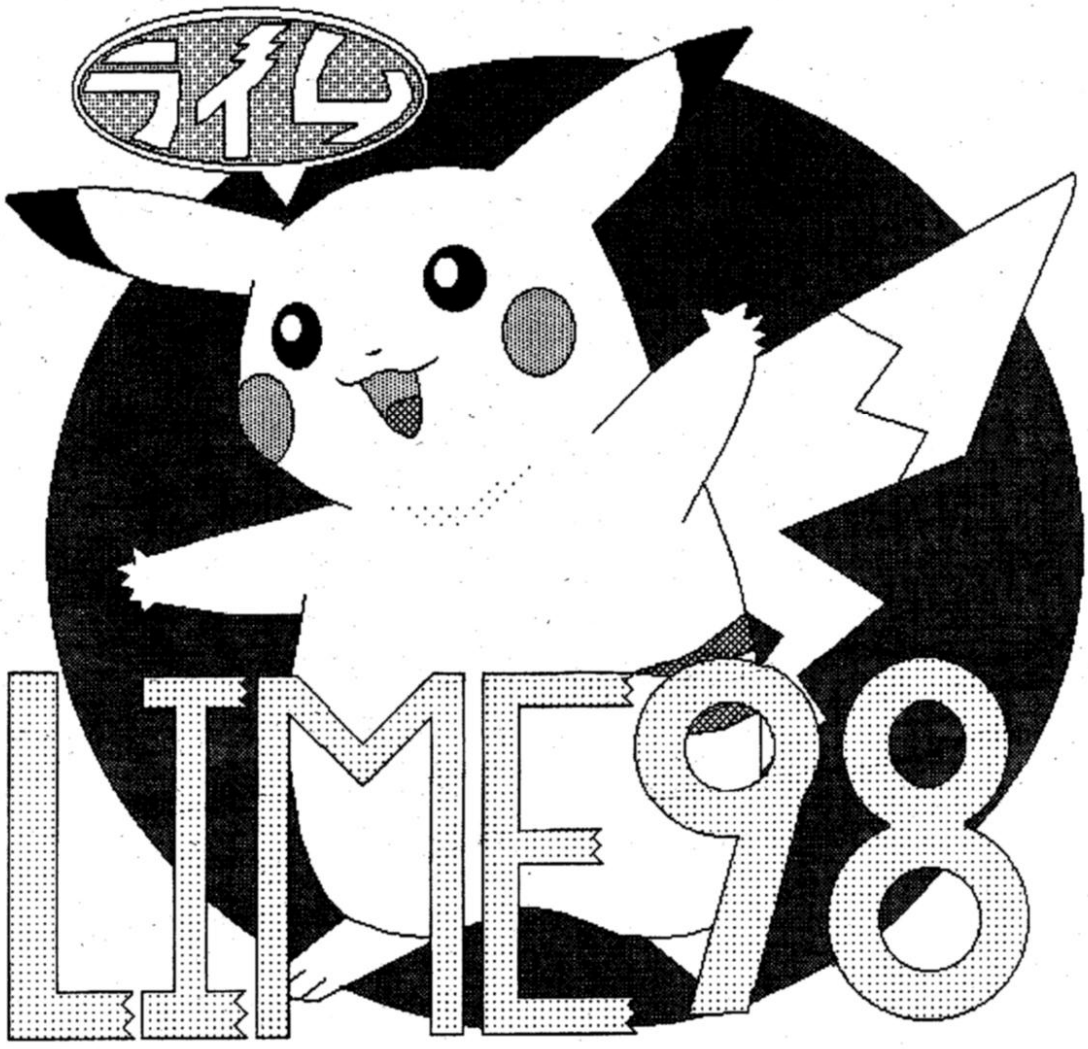


Limited Expression Reports

K.I.T. コンピュータクラブ

あたくしは圧倒的に吉川ひなののファンだ

たけおか@AXE

1

あたくしは、一色紗英がかなり好きだ。中山エミリもだいぶ好きだ。山口紗弥加なんぞも好きだ。だが、圧倒的に吉川ひなののファンだ。

なぜなら、あたくしとひなのは、誕生日がおんなじだからである!!!!(これはびっくりマークを 4 つぐらいつけて強調せねばなるまい) 美少女と誕生日がおなじとわ!

さて、しかし、あたくしは秋吉久美子などという、今では、おばはんと言わざるを得ない人も圧倒的に好きだ。紺野美沙子も好きだし、黒木瞳も好きだし、鈴木保奈美だって好きだ。

でも、鈴木保奈美は、なんで? なんで、こいつ鈴木保奈美と結婚すんだよ、というべき、ダメな男と結婚した。石橋なら、F1-メカ-ヲタクのカワイちゃんの方が 1000 倍ぐらいマシじゃねーか。

2

あたくしは、DEC Alpha が好きだ。
Intel は嫌いだ。
Motorola はもっと嫌いだ。
でも、PowerPC は IBM オリジンだから許そう。
Sparc は論外に嫌いだ。
Mips が好きだ。

最近、実家で MicroVAX が稼働するようにした。SUN-3 も復活させた。Symbolics3600 も稼働するようにした。本当に本物の高級言語マシンだ。普通の家で、MicroVAX と Symbolics と SUN-3 が稼働する家は、そうはあるまい。これで、CIT の縦長キャラクタ端末をつければ、AI 開発環境は完ぺきだ。おーほっほっほ。(DEC 2020 は手に入れ損ねたが) ま、こうしてあたくしの実家では、CISC の代表が一同に会しているのだ。HP-700 も金五百円也で買ったしな。:-)

でも、CommonLisp を動かすなら普通の X86 マシンの FreeBSD 下で Emacs と GNU Common Lisp を使った方が、高速で快適だ。;-<

一体、世の中というものには、夢も希望も無くなってしまったのだらうか?

3

あたくしは、DEC Alpha のクズも集めている。ジャンクの DEC 3000 は 3 万円で買った。HDD もメモリも付いてた。AlphaStation200 というローカル FM 局の様な名前の機械も買った。ま、それがどうしたといふのだろう。CISC の夢が無くなった現在には、DEC Alpha で世界最高クロックの夢を見るしかないではないか。

4

JavaChip というたわけがある。
Stack マシンが速くなりゃ、誰も苦労せんて。
SUN は Sparc は作るわ、JavaChip は作るわ、で、世界をダメにする酷いやつらだ。
SUN はアーキテクチャ設計やめろよな。

5

Java 仮想マシンと、Java コンパイラがある。あたくし達は、1981 年の学部 2 回の時にまったく同じ発想をしていた。ネットワークでバイナリを送り込むことも考えていた。というより、ネットワークがあるから、バイナリ共有をしようと考えた。これも Java と同じ。1981 年 11 月の Lime には、はっきりとそのことが記してある。しかも、「Core」と呼ばれる仮想マシンは Stack マシンとしていた。Stack マシンは阿呆でも実装できるから。

1981 年、当時、マイコン (パソコンという言葉はなかった) には、メモリもなく、フロッピーディスクもほとんどなく、コンパイラが動作する機械は少なかった。

そこで、コンパイルは 1 度だけ、そしてバイナリはどこででも動く。Compile Once, Run Everywhere だ。

バイナリはネットワークを通じて共有し、どんなアーキテクチャを持つ機械でも同じプログラムを実行できるようにしよう。と、考えていた。

でも、あたくしは、その構想を途中で捨てた。当時の機械は遅すぎて、どうにも面白くなかったから。

PC8001(Z80 4MHz) の上に Core を実装した偉い後輩もいたが。

だから、Java が出た時、笑った。
あまりに同じすぎたから。
ゴースリングはマイコン小僧で、あたくしもマイコン小僧だ。
マイコン小僧の考えることは万国共通だ。
しかも時代を越える。

6

あたくしは、SUN が時代に乗って成功し、あたし達は学生だったから負けた、などと、いうつもりは毛頭無い。そもそも負けている気がしない。

ただ、マイコン小僧はどこでもいつでも同じだ、と、言いたいだけ。

あたしの書いたプログラムや、作ったシステムは、いつかどこかでマイコン小僧の目に触れ、「ああ、こういうこと、俺も考えてたよ。ニコッ」と言われるだろう。

物作りは楽しい。
システム設計は楽しい。
アーキテクチャ設計は楽しい。
プログラミングは楽しい。
半田づけは楽しい。
ラッピングは楽しい。
デバッグは楽しい。
オシロを見るのは楽しい。

さぁ、今日も 17 年後に流行るものを考えよう。
まっ、最近、そういう 不思議少女 ひなの的 時空超越的な気分なのである。

妄想卒業研究

電子情報工学科 知能制御研究室

野村 實

[1] ざっくばらん

私もいつのまにやら４回生になってしまった。今頃になって電子回路なんぞに手を出している。中学生のころ通信教育で（はんだ付けするだけ）やったのが最初だろうか。当時はＩＣは謎のゲジゲジ（何かスゴイもの）だったものだ。今見ればただの CMOS の NOR だった。抵抗とコンデンサ間に入れて入力短絡してる。一階線系微分方程式で解くと 400Hz 程度の発信回路とわかる。今回は知能制御研究室にちなんで制御関係に手を出した。（トランジスタを使ったモータの制御など）AXE で集中砲火をあびながらも一人で色々作れるようになった。DC モータをトランジスタでスイッチングする時の問題点にモータの制止、回転方向の切替え、等が挙げられる。モータを正確に停止させるためには、トランジスタのスイッチングの改善、モータの逆起電力の抑止などが必要となる。具体的にはベース電流を必要以上に流さないよう適当なベース抵抗を入れることで素子の蓄積効果の軽減をはかる。さらに、抵抗に並列にコンデンサ（スピードアップコンデンサ）を入れ立ち上がり時間を短くする。又、逆電流についてはダイオードで整流する。

2 プラント（モータ 2 つ）で車を動かす場合、各プラントのブリッジの hFE の差が問題となる。電流増幅率の差はモータの回転数の差となり正確に前進後退させることが困難となる。コンプリメンタリを用い、さらにそのランクも等しくすることで大幅に改善されるが、実際にそのような素子をそろえるのは至難だ。

今回制御を試みたマニピュレータは 5 プラント、2 リンクのマニピュレータである。コンプリメンタリを 10 組（ランクはばらばら T_T）そのドライブにさらに 10 個の PNP トランジスタを用い合計 30 個のトランジスタでモータを動かしている。このぐらいの数になってくると値段が気になってくる。今回見て回った限りでは 2A 程度流せるもので 50～120 円、3A になると 100～200 円といったところだった。ちなみに 2SC1815、2SA1015 などのメジャーなものは 20 円。

今回のような電流駆動をする場合、各抵抗の役目を把握してないと素子を潰してしまう。あまり頭使いたくない人は MOS-FET を使った電圧駆動がおすすめ（値段は謎）。

[2] ニューロコンピュータ

FSA（決定性有限オートマトン）のニューラルネットワーク表現及びその同定法が今のところの研究テーマだ。全ての素子がニューロンからなるニューロコンピュータは LAN などのネットワークへの接続は考えにくく、強力なフェールセーフ機構として防衛、原子力などのプロセス制御、等への応用が期待されて

いる。

私自身は現在主流である計算機のための新たなプロセッシングユニットとして FPU などと同列のものと考えている。言語処理、パターン認識、連想記憶などの処理に用いる。

FSA を現在の計算機上に実現するとテーブルの形になる。言語処理のための大規模な FSA となるとテーブルのパラメータは非常に多くなり作成困難である。２次ニューロン（前状態の出力を入力へフィードバックしたもの）を用いると FSA を入出力データから学習によって自動生成することができる。課題は学習方法（アルゴリズム）の決定とその収束の論理的証明にある。

これは今回の様なマニピュレータを制御するのにも応用ができる。マニピュレータの動きは状態遷移ととらえることができ、FSA で表現可能である。入出力パラメータから FSA を同定し禁止遷移を排除することができれば、遷移テーブルを作成する必要はなくなる。

[3] あとがき

月日の過ぎるのは早い。卒業間近になると感じる。４年前を振り替えってみると大学生活で得たいと思っていたものは結構得られた様に思う。卒業までの残り数ヶ月、頑張ればもう一押し（この辺の一押しの有無で随分差がでる）出来るかもしれない。（サボったままで理解不十分で放置してる大学の履修科目とかもキッチリしとかなあかんなあ）

自己評価ではあるが現在、４年前とは比較にならないレベルに達したという実感はあるしかし、上を見ると残念ながらそこがまだまだレベルの低い所である事を知る。修行の道は長い。

おもえば部活動のに随分時間を浪費した。得るものが無かっただけでなく怨みをかった。無念だ。（ぐち:-P）

マズイのは技術的な問題をクリアしている分野において、面白い事があまり出来ないまま卒業の時期を向かえてしまった事だ。はたから見れば色々結果が出ているのだろうが、勝手に出ているだけでここで言う面白い事に相当しない（本質的に自分自身を満足させるものでない）

学祭で面白い事が出来ればよかったのだが時間的にイマイチで終りそうだ。一応、研究室での修行をいかした理論的にしっかりしたフィードバックが目標。研究室によるとは思うが４回生になると自由に使える時間が減る。単位のあまりそろってない３回生は次の春の長期休暇のうちに何かやっとかな４回生で学祭に参加するのはしんどくなるかも。

今後のコン部の行方や如何に？

FreeBSD で PIC 開発環境を揃えるのだダダダ!!

電情三回 安達 浩次

1 はじめに

PIC とはマイクロチップ・テクノロジー社のワンチップ・マイコンなのだ。簡単に言えば、ワンチップ・マイコンとは一つのチップの中に CPU、RAM、ROM そして I/O をそなえたもののことを指すのだ。

その中でも今回特集する PIC はプレステの MOD-Chip やサターンのサターン・キーにも採用された、ある意味、人気の高い、有名なチップなのだ。もちろん、そういった暗黒面だけでなく、つぎのような利点をもった非常に優秀なマイコンなのだ。ホンマ?

- 開発環境が安価に揃う。

って、一つしか思いつかんかった。しかし、これにつきるであろう。これは、PIC のプログラム・ライタが自作できるということと、PIC の開発元であるマイクロチップ・テクノロジー自身がアセンブラとリンカを無償で公開していることに由来する。もちろん、PIC 自身が数百円であることも重要であろう。

開発環境が安価に揃うということは貧乏学生の遊びには非常に向いているのだ。

今回はそんな PIC の開発環境を FreeBSD で揃えてみたのだ。実は後にしめすホーム・ページにその情報のすべてがあるのだ。だから、実はこの原稿の情報は GNUPIC のホーム・ページの URL しかないのでした、ちゃんちゃん。しかし、ターゲットは Linux だし、使ってみないことには分からないので、まとめてみたのだ。したがって、今回は DOS や Windows を用いた開発環境はしらないのだダダダ。

(注) また、今回は PIC の中でも PIC16C84 と PIC16F84 について限定した話なのだ。なぜなら、それ以外はわからないからなのだ。

2 PIC 開発の流れ

PIC の開発にはなにが必要なのかなのだ。

大雑把にいって、PIC の開発は以下のように行なうのだ。(そりゃそうだ。)

プログラミング → アセンブル → 書き込み

したがって、とりあえず PIC の開発に必要になるのは以下の 3 つなのだ。

- アセンブラ
  いきなり、オブジェクトコードを書くのはちょっと大変なのだ、ニモニックで書きたいのだ。したがって、アセンブラを用意するのだ。

- ライタ
  アセンブラで作成した、オブジェクト・ファイルを PIC に書き込むための装置のことだ。電圧レギュレータとトランジスタ、3-state バッファからなるのだ。これに関しては、インターネット上に多数、回路図が存在するのでそれも参考にするのだ。

- ライタ・ソフト
  ライタをコンピュータから扱うために、ソフトが必要なのだ。FreeBSD で動くライタ・ソフトは存在しないので、それだけはどうにかするのだ。

3 実際の話なのだ。

・アセンブラ

gpasm

実は、UNIX で動作する PIC のアセンブラは以外に多数あるのだ。その中でも比較的使いやすいと感じたのは gpasm だ。独断と偏見なのだ。現在のバージョンは 0.0.6 Alpha だ。なぜ使いやすいかというと、gpasm はマイクロチップ社謹製の MPASM に真似て作られているからなのだ。したがって、トラ技などに載っているサンプル・プログラムがそのまま動作する可能性があるのだ。やはり書式や疑似命令などが同じなのはかなりうれしいのだ。

gpasm も GNUPIC のホーム・ページでゲットできるのだ。

gpasm-0_0_6_tar.gz をセットアップするだ。

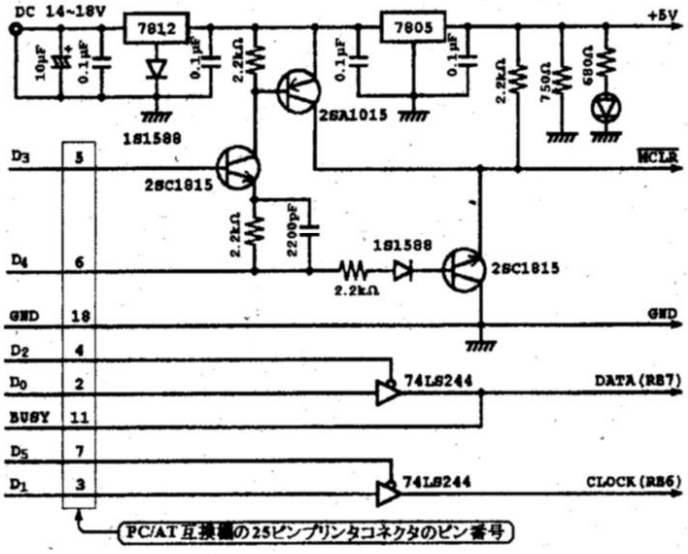
make して使えるようにするのだ。 configure ファイルが入っているので簡単だ。

```
% tar xvzf gpasm-0_0_6_tar.gz
% cd gpasm-0.0.6
% ./configure
% make
# make install
```

これだけの作業で使えるようになったぞ。**/usr/local/bin/gpasm** だ。

・ライタ

先に装置の方がないとしかたがないので装置を作成するのだ。先にも述べたが、PIC ライタ自体はレギュレータ、トランジスタ、3-state バッファからなるだけなのだ。今回、私が作成した PIC ライタの回路図を以下に示すのだ。



作成は簡単なのだ。難しいのは D-sub コネクタの基盤への取り付けと TEXTOOL の IC ソケットを購入する決心をするところだろう。(なぜ、この IC ソケットが 2000yen 以上も)

動作確認のため Windows 用の書き込むソフトで書き込めることを確認するのがよいのだ。(私はしなかったのだ)

・ライタ・ソフト

先ほど作成した PIC ライタのための書き込みソフトを用意するのだ。PIC 書き込みのためのアルゴリズムはマイクロチップ社により公開されており、自分で作成することができるのだ。しかし、一から作るのも大変だなーと思うのだ。

GNUPIC のホームページにはさまざまな書き込みソフトが用意されているのだ、しかし Linux 用だったりするので、FreeBSD 用かつプリンタ・ポート接続タイプ用の書き込みソフトを作るのだ。もっとも、そのころは GNUPIC のホームページのこと

などは知らなかったのだ。

と、いうわけで、アセンブラとライタ、ライタ・ソフトと揃ったのでとりあえずPICの開発はできるようになったのだダダダダ。

4 実況生開発

これまでで、とりあえずPIC開発はできるようになったので、実際の実況生開発とGNUPICのホーム・ページで公開されていたその他のツールの使用感を報告するのだ。

・サンプル・プログラムを入力

リスト1のプログラムはLED二つを1秒間隔で点滅させるプログラムだ。実はこのプログラムもトラ技のホーム・ページで入手することができるのだ。ただし、もとのサンプル・プログラムで使用されていた疑似命令DEはgpasmではDATAなのでそこだけ変更しているのだ。

とほうにくれていると…

DOS 用の picwr.exe という書き込みソフトがソース付きで配布されていたのだった。そこでこれをもとに FreeBSD 用を作ろうと思ったのだ。　そこで問題となったのが、FreeBSD で直接 I/O をさわるにはどうすればよいのかであった。真剣に作るとなると FreeBSD みたいなマルチ・タスク OS ではちゃんとデバイス・ドライバを作成してカーネルが制御を行なうべきであるが、私のスキルの問題と時間の問題によりとりあえず動けばよいであろうと考えたのだ。そして、デバイス・ドライバの開発はやめといたのだ。

しかし、それはそれで問題が…

というわけで、普通のコマンドとして動く書き込みソフトを作成することにしたのだ。しかし、そこでも問題が…。普通のプロセスで I/O にアクセスするのってどうするの? 実はこれに関しては見当がついていたのだ。　その昔、SoftwareDesign 誌に FreeBSD の lpt ドライバに手を加えて ioctl のコマンドを追加し、任意のデータを出力できるようにしたサンプルの記事がのっていたのだ。そしてすでに、記事のとおりドライバに手をいれて実験は行なっていたので、勝利の確信はあったのである。

しかし、うごかないのだダダダ。

動かなかった、しかも理由がいまだにはっきりしていない。とにかく良く分からずに動かなかったのだ。　結構、気合いをいれていたが、動かなくなるという一度の挫折でいつもの悪い癖が出てきて、かなりあきらめていたのだった。

しかし、自体は 3 転。

動かないという話を駒嵐さんにしていると/dev/io に関する話がでてきた。そうである/dev/io を忘れていたのである。しかし、私は/dev/io の使い方を知らなかったのだ。でも、調べてみると簡単。プロセスが/dev/io をオープンすると、その後は自由に inb や outb が使用できるようになるのである。こりゃ便利なのだ。　そこで、先ほどの DOS 用の picwe.exe のソース picwr.c の中で inb と outb に相当するところと PC 内部のタイマを利用して作成していた usleep に相当する関数をそれぞれ置き換えてやると。ななんと、バッチリ動いたのさ!!

リスト１１/3

```
;       PIC test program
;
        LIST    P=16C84
;
;       PIC16C84 Register
;
INDF    EQU     0
TMR0    EQU     1
PCL     EQU     2
STATUS  EQU     3
FSR     EQU     4
TRISA   EQU     5
PORTA   EQU     5
TRISB   EQU     6
PORTB   EQU     6
EEDATA  EQU     8
EECON1  EQU     8
EEADR   EQU     9
EECON2  EQU     9
;PCLATH EQU     A
;INTCON EQU     B
; STATUS Register bit
C       EQU     0
DC      EQU     1
Z       EQU     2
PD      EQU     3
TO      EQU     3
RP0     EQU     5
RP1     EQU     6
IRP     EQU     7
; OPTION Register bit
PS0     EQU     0
PS1     EQU     1
PS2     EQU     2
PSA     EQU     3
T0S     EQU     4
T0CS    EQU     5
INTEDG  EQU     6
RBPU    EQU     7
; INTCON Register bit
RBIF    EQU     0
INTF    EQU     1
T0IF    EQU     2
RBIE    EQU     3
INTE    EQU     4
T0IE    EQU     5
EEIE    EQU     6
GIE     EQU     7
```

```
; General
TRUE    EQU     1
FALSE   EQU     0
YES     EQU     1
NO      EQU     0
;
DELAYT  EQU     0D
TIMEF0  EQU     0F
TIMEF1  EQU     10
TIMEF2  EQU     11
B0      EQU     00
B1      EQU     01
;
;********* Program Memory Programming ***********
;
        ORG     00
        BSF     STATUS,RP0          ;Select page 1
        MOVLW   0FCH                  ;Set Input/Output
        MOVWF   TRISA                  ;SET PORT A
        BCF     STATUS,RP0          ;SELECT PAGE 0
LOOP
        BSF     PORTA,B0          ;Output bit0=H
        BCF     PORTA,B1          ;Output bit1=L
        CALL    DELAY
        BCF     PORTA,B0          ;Output bit0=L
        BSF     PORTA,B1          ;Output bit1=H
        CALL    DELAY
        GOTO    LOOP
;
DELAY
        MOVLW   DELAYT
        MOVWF   TIMEF0
LOOP0
        CLRF    TIMEF1
LOOP1
        CLRF    TIMEF2
LOOP2
        DECFSZ  TIMEF2,1
        GOTO    LOOP2
        DECFSZ  TIMEF1,1
        GOTO    LOOP1
        DECFSZ  TIMEF0,1
        GOTO    LOOP0
        RETURN
```

リスト 1 3/3

```
;
;********* EEPROM Data Memory Programming **********
;
        ORG     2100H
        DATA    "ABCD"
        DATA    1,2,3,4,5,6
;
        END
```

・アセンブルなのだ

さっそく、作成したプログラムをアセンブルするのだ。

```
% gpasm hello.asm
```

BOX のマシンは高速なのだ。一瞬で hello.hex は出来上がっているのだ。中身をみてみよう。

```
% cat hello.hex
:100000008316FC3085008312051485100B20051023
:1000100085140B2004280D308F0090019101910B65
:0C0020000F28900B0E288F0B0D280800F5
:1042000041004200430044000100020003000 4009A
:04421000050006009F
:00000001FF
```

おっ、なんからしいものが出来ているのだ。これを書き込めばきっと出来るのであろう。

・ライタ・ソフトを動かす

PIC ライタの電源をいれて、PC に接続するのだ。そして

```
% picwr
```

と入力するのだ。てててて、実行のログがとりたかったのだけど、実は開発をしていたノート PC を修理にだしていてサンプルもなにもかも手元にないのです。本当はもう一度つくってでもログをのせるべきなんだけど…。学園祭直前でしかも〆切もすぎているので、この辺で勘弁してちょ。

あとがき (ロボットの開発も含む)

今回も例によって余裕がなかった。本当は余裕があればロボット開発に使用している日立の H8 について書きたかった。しかし、いま原稿を書いている時点では H8 に関しては全然手が回っていない。現状は秋月のキットを組み立てて、サンプル・プログラムで動作確認をした程度である。したがって、FreeBSD を用いて開発環境を整えるのはかなり先の話である。

実際に作業を始めると根本的な知識の欠如を痛感させられた。学園祭前には BOX で作業をおこなっていたのだが、これまでの「いつもまわりに知識をもった人達がいる。」という環境をはなれると本当に基本的なことをするのでさえかなりの時間がかかることがわかった。

その点に関しては野村さんにいわれたとおり、もっと日頃からクンフーを積むべきであったと思う。反省。

あーあ、ほんまは「URISC 計画」を展示するつもりだったのだけどなー。

ＭＪゲームプログラミングの構造と実行(ウソ

かねとみつゆき

そういえばとある人が麻雀ゲームを作るとか言ってたなぁ～
話を聞いてみると役の判定とコンピュータの思考部分に困ってるらしい。
ほ～、じゃ、私がやってみたろうじゃないか。

（１）役の判定

例としてピンフをあげてみると
鳴きがなしで順子が３つ、対子が１つに両面待ちの状態。対子の２枚は
場風、自分の風、三元牌であってはいけない。
この状態をソースで表すと

```
/* ピンフ */


if(
        (NakiList->[count]==0)
        &&(HaiTbl[shuntsu]==4)
        &&(HaiTbl[toitsu]==1)
        &&(CategoryCountTbl[kaze]==0)
        &&(CategoryCountTbl[sangen]==0)
        &&(MatiList[ryanmen]==1)
)
```

制御的にはこんな感じでしょうかな。
NakiList->[count]ってのは構造体、下に count（鳴いた回数）、ti(チーした回数)、pon（ポンした回数）とかを持ってもいいんとちゃうかな。
HaiTbl[?]はあがりの手牌から shuntsu なら順子、toitsu なら対子、anko なら暗刻などを数を数えていれてます。
CategoryCountList はあがりの手牌から風牌 kaze、三元牌 sangen の数が入ります。
MatiList[?]は待ち牌の種類を？で表した変数 tanki, ryanmen, etc.. にいれます（待ち可能であれば１、不可なら０とか）。

こんなふうに上がりの形ってのは
対子、暗刻、槓子、順子で分けてしまえば判定は楽なはずだ！

その分け方や判定方法にもちょっとコツがあって下のようにすると
結構重ならずに判定できる。

１－２）役・判定法

まず、あがり形の牌を取り種類別に数を数える。
そうすると下のいずれかのパターンの集まりになる。

①２の倍数
②３の倍数
③３の倍数＋２

もし①の組が７つできた場合、国士無双か七対子が考えられる。

この二つを比較したい時は２枚ずつ差をとってやればいい。
差が０になるようなら、七対子になる。

それ以外の役については下の判定にかければよい。
まず、あたり牌を除いた１３枚に対しソートをおこない、種類ごとに２枚ずつ下の比較を行う。

０）左から２枚取る。（小さいほうから２つ）。

１）右の牌から左の牌の差を求める（右の牌の方が大きい数です）。

差が「２」ならカンチャン待ち。
「１」ならつぎへ進む。
「０」なら３）へ進む。

２）先ほどの右の牌ともう一つ右の牌の差を求める

差が「１」なら順子
それ以外で左の牌が１か右の牌が９ならペンチャン待ち
上の２つにあてはまらない場合は両面待ち

３）先ほどの右の牌ともう一つ右の牌の差を求める

差が「０」なら暗刻
それ以外なら左の２つは対子となる。この場合は新しく取ってきた牌とその右の牌を０）で取ってくる。

以上を一通り終わるまで続ける。
これで暗刻、対子、順子、と待ちはわかるはず。

これから国士無双、七対子以外の役のチェックを行う。

1）ピンフのチェック

これについては前に書いたように判定してもらえばよい。

2）トイトイ系のチェック

・トイトイ　1回以上の鳴きが行われ暗刻３つと対子で構成されている場合。
・三暗刻　暗刻（鳴きの行われない）が３つある場合
・四暗刻　暗刻（鳴きの行われない）が４つある場合
・三槓子　槓子が３つある場合
・四槓子　槓子が４つある場合

3）タンヤオのチェック

すべて1・9・字牌以外で構成されてる場合

4）チャンタ系のチェック

構成された面子すべてに1・9・字牌が含まれる場合

・清老頭　すべての牌が1・9牌である
・混老頭　すべての牌が1・9・字牌である
・ジュンチャン　すべての面子に1．9牌が含まれる
・チャンタ　上の３つ以外、構成された面子に1・9・字牌が含まれている

5）チンイツ系のチェック

・ホンイツ　字牌とそれ以外の一種類で構成されている
・チンイツ　字牌以外の一種類で構成されている
・字一色　字牌一種類で構成されている

6）字牌系のチェック

・翻牌　３枚以下の風牌、２枚以下の三元牌がある
・大三元　三元牌の暗刻の数が３枚ある
・小三元　三元牌の暗刻の数が２枚あり、残りの三元牌が頭である。
・大四喜　風牌の暗刻が４枚ある
・小四喜　風牌の暗刻が３枚に残りの風牌が頭である

では、以上の条件を実際に条件分岐してみよう。

```
void CheckPinfuToitoi
(int Player, int PlayerCheckListVal, bool RonAgari)
{
    if ( /* ピンフの判定　1）部分*/
        (PlayerNakiList[Player]->Count == 0) &&
        (MatiCountTbl[PaiToitsu] == 1) &&
        (MatiCountTbl[PaiSyuntsu] == 4) &&
        (SrcMatiCountTbl[PaiRyanmen] == 1) &&
        (CategoryCountTbl[PAICATEGORY_KAZE] == 0) &&
        (CategoryCountTbl[PAICATEGORY_SANGEN] == 0)) {
        SetYakuList(YakuPinfu, YakuTbl[YakuPinfu].Han);
    }
else if (/* トイトイの判定　2）部分　*/
        (MatiCountTbl[PaiToitsu] == 1) &&
        (MatiCountTbl[PaiSyuntsu] == 0)) {
```

```
        if (
            (MatiCountTbl[PaiKoutsu] + MatiCountTbl[PaiPon] +
             MatiCountTbl[PaiAnKan] + MatiCountTbl[PaiMinKan] >=
4)) {
            SetYakuList(YakuToitoi, YakuTbl[YakuToitoi].Han);
        }
    }
}

void  CheckSananko
(int Player, int PlayerCheckListVal, bool RonAgari)
{ /* 暗刻系の判定　2）部分　*/
    int CheckAnkou, CheckKanko;

    if (YakuList->Count != 0) {
        if      ((int)YakuList->Objects[YakuList->Count-1]      ==
HanYakuman)
            return;
    }
    if (!(MatiCountTbl[PaiToitsu] == 1) && CheckNotMatu(0))
        return;
    CheckKanko = MatiCountTbl[PaiAnKan] + MatiCountTbl[PaiMinKan];
    if ((MatiCountTbl[PaiToitsu] == 1) && (CheckKanko == 4)) {
        SetYakuList(YakuSuKantsu, YakuTbl[YakuSuKantsu].Han);
        return;
    } else if ((MatiCountTbl[PaiToitsu] == 1) && (CheckKanko == 3))
{
        SetYakuList(YakuSanKantsu, YakuTbl[YakuSanKantsu].Han);
    }
    if (RonAgari)
```

```
        CheckAnkou      =       SrcMatiCountTbl[PaiKoutsu]      +
MatiCountTbl[PaiAnKan];
    else
        CheckAnkou      =       MatiCountTbl[PaiKoutsu]         +
MatiCountTbl[PaiAnKan];
    if ((MatiCountTbl[PaiToitsu] == 1) && (CheckAnkou == 4)) {
        SetYakuList(YakuSuAnko, YakuTbl[YakuSuAnko].Han);
    } else if ((MatiCountTbl[PaiToitsu] == 1) && (CheckKanko == 3))
{
        SetYakuList(YakuSanAnko, YakuTbl[YakuSanAnko].Han);
    }
}
```

ページ数の関で判定ルーチンが途中までしか書けまへんでした。
残りを聞きたい人は私まで来てちょ。

（2）コンピュータの思考部分

次にそいつが困ってたのが思考部分。
牌をつもってランダムに切るのなら誰でもそんなプログラムは作れる。
切るときに「どんな手を作ろう」とか考えなきゃいけないのがつらいとこ。
で、実際、コンピュータにそこまでさせるのはふつーの人には無理だ。
＃周りにはそーゆーことを専門にやってる人もいるけど…
何順かランダムで切ってみてある確率で山に残ってる牌と自分の手牌で役を作って

あがるってのもいいけどそれじゃ、うさんくさい。

まず、配牌を調べて一番あがりやすい手を考えるのが妥当だろう。

（1－2）の0）から3）で表した。判定ルーチンで待ちや面子が確定するので
それに必要ない牌を切るようにプログラムしたらいいかね。

今回、あまりプログラムについてはふれられなかったのが悲しい。
題名もホントにうそになってしまってるし
実際動くようになるには結構時間がかかりそうだし…
＃今年の出展には間に合わなかったし。
来年にはそのルーチンごとプログラムを出展したいなぁ～

ISAのカード、作ってみませんか?

電情3回 服部 保

0 前提とお願い

この記事は、ある程度電子回路が分かる人向けに書かれています。「そんなん知らんわ」という方は、お読みにならない方が賢明かもしれません。お詳しい方は、この記事の中に含まれている (であろう) 筆者が気づいていない間違いにお気づきになるかもしれません。そういった点に関しては、お知らせいただければ幸いです。

注: この記事は原則無保証です。資金的問題、および部品の調達に時間を要し過ぎた (A/D コンバータが取り寄せの上に納期不明) ため、作動検査を行うところまで出来ませんでした。このため、回路には大嘘が含まれている可能性があります。

1 はじめに

ハードウェアなんかをちょっとかじったりすると、自分で「PC の拡張カードとか作れたら面白いなぁ」なんて思うかもしれません。この話は、そんな人にとってほんのちょっとだけ役に立つかもしれません。

「PC(ここでは AT 互換機を想定する) の拡張カード」と一口に言っても、まず規格からしていろいろあります。これらが挙げられるでしょう (というより、これ以外ない)。

- XT バス …… 8 ビット幅。かなり昔からある。
- ISA バス …… 16 ビット幅。PCI に押されて滅びそう。
- PCI バス …… 32 ビット幅。現在の標準的なバス。
- AGP バス …… 32 ビット幅。グラフィックカード専用。

これらのうち、グラフィックカード専用の AGP バスは無視するとして、作りやすさなどを考えて一番妥当なのは ISA バスでしょう。PCI バスのカードを作ろうと思うと、かなりしんどい目を見ることは火を見るよりも明らかです。なにしろアドレスバスからデータバスまで 32 ビット幅なので、まず回路がめちゃくちゃ複雑になります。また、バスクロックが 33MHz と高速なので、回路をちゃんと考えないと遅延やら何やらでいやな目に遭いそうです。

その点、ISA バスはアドレス、データともに 16 ビット幅で、バスクロックも (PCI よりは) 遅いので、PCI バスよりも楽になるでしょう。

XT バスはアドレス 16 ビット、データ 8 ビットなので、ISA バスよりもさらに作りやすいでしょうが、いかんせん 8 ビットでは使いでに欠ける感があります。

というわけで、ISA バス用のカードを作るにはどうしたらいいのかの基本的なことについて見ていきたいと思います。ISA バスは XT バスを 16 ビット幅に拡張しただけなので、ISA バス用カードと同じ要領で作れるでしょう。

2 ISA バスについて

ISA バスのコネクタピンと信号名の関係を図 1 に示します。SAxx はアドレスバス、SDxx はデータバスです。上に $\overline{}$ が付いているものは、アクティブ・ロー (L レベルで機能がアクティブになる) です。

ISA バスと CPU の間でデータをやりとりするときは、次のように動作します。

- 書き込み時
  1. データを出力するポートを指定するためにポートアドレスをアドレスバスに出力する
  2. データをデータバスに出力し、書き込み信号 $\overline{\text{IOW}}$ を出力 (L レベルに) する
- 読み込み時
  1. データを入力するポートを指定するためにポートアドレスをアドレスバスに出力する
  2. 読み込み信号 $\overline{\text{IOR}}$ を出力し、データ入力を行う

3 こんな部分が要ります

実際にカードを作る段で必要なのは、以下のような部分です。

- チップセレクト信号生成部
- IOCS16 信号生成部
- 機能部

ここで、チップセレクト信号というのは、ある機器だけをアクティブにするための信号です。FreeBSD にしろ Windows にしろ、ある機器を増設したときの設定項目に IO ポートアドレスというのがありますが、これがまさにそれです。アドレスバスに、

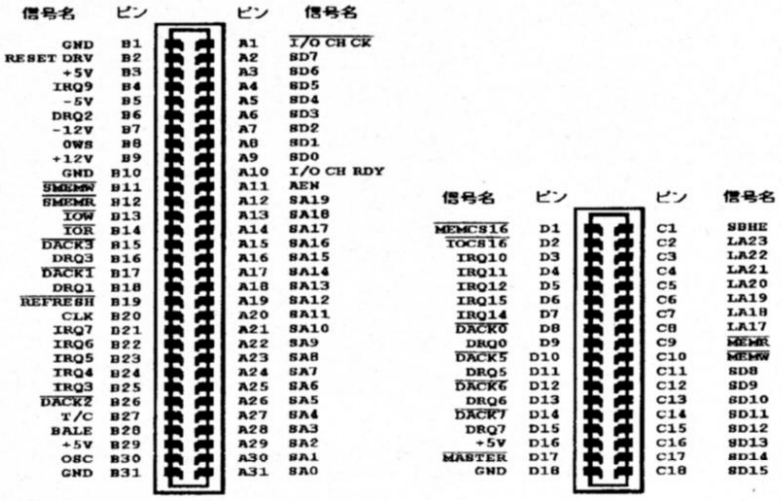


図 1: ISA バスのコネクタピンと信号名

使いたい機器の IO ポートアドレスを流すと、その機器のチップセレクト信号だけが生成されて、機能がアクティブになる、という仕掛けになっています。

また、IOCS16 信号というのは、ISA バスを 16 ビットバスとして動作させるための信号です。なぜこんな信号があるかといえば、もともと ISA バスは 8 ビットの XT バス (62 ピンのスロット) に、16 ビットアクセスが出来るようにデータ線や制御線を追加した 36 ピンのスロットを足して作られたため、8 ビットのカードと 16 ビットのカードのどちらを差しても動作するようにする必要があったからです。実際、ISA バスでは、IOCS16 信号がカードから出力されていないときは 8 ビットバスとして動作するようになっています。もっとも、最近は 8 ビットのボードはほとんどないので、IOCS16 信号の意味はほとんどないようにも感じますが (IO データ機器から出ている IF-SEGA/ISA というセガサターンのパッドを PC に繋ぐためのボードぐらいしか見かけません)。

機能部というのはそのままです。要するに温度計なら温度を計測する部分、モータ制御カードならモータ制御部分です。

4 回路を考えよう

以上のことを踏まえた上で、「ISA バスに差す温度計」の回路を考えてみました。

まず、カードの仕様を決めます。ここでは、こんな感じにしました。

IO ポートアドレス	0x0300H〜0x03FFH のうち連続する 2 つ
IRQ	なし
温度計	8 ビットで量子化、0〜127°C まで 1/2°C 単位

温度計については、あまり高精度にしてもしょうがないので、8 ビットにしました。12 ビットぐらいにすると、1/16°C 単位で計れたりするのですが、12 ビットの A/D コンバータは高くつく (大体 2000 円以上) ので、製作費の面からも 8 ビット程度が手ごろです。

IO ポートアドレスについて補足しておくと、16 ビットバスとして使うときは IO ポートアドレスは偶数番地から始める決まりになっていますので、実際に指定できるのは 0x0300, 0x0302, ……, 0x03FE の 128 個です。

あと、IRQ はなしにしましたが、これは筆者の勉強不足のせいです。

4.1 チップセレクト信号生成回路

回路図は図 2 のようになります。チップセレクト信号は、アクティブ・ロー (信号が L レベルのとき機能 ON) なので、普段は H レベルの信号を出力させるようにします。74HC688 は 2 つの 8 ビット 2 進数の等・不等を比較するものです (8bit equal-to comparator)。これは、Q_0〜Q_7 にディップスイッチで作った状態と同じ入力が P_0〜P_7 にあれば、$\overline{P=Q}$ に L を出力します。それ以外のときは H が出力されます。ここでは、アドレスバスの下位 8 ビット (ただし、IO ポートアドレスの指定は偶数番地からなので、最下位ビットは 0 に固定しています) を見ています。上位 8 ビットは 4078 で見ています。SA15〜SA8 に 0x03H が入ると L が出力されます。これらと BALE 信号とから、チップセレクト信号を作っています。つまり、74HC688 の出力:L、4078 の出力:L、BALE:H のときだけ L が出力されるようになっています。

なお、BALE 信号は、ISA バスと CPU がつながっているかを示す信号で、BALE:H のときにつながっていることを示します。

4.2 IOCS16 信号生成回路

これは、非常に簡単です。74LS125(3state Bus-Buffer) を用いて、普段は信号出力なし (ハイインピーダンス状態) で、チップセレクト信号が来たときだけ L が出力されるように作ってやるだけです。図 3 のようになります。

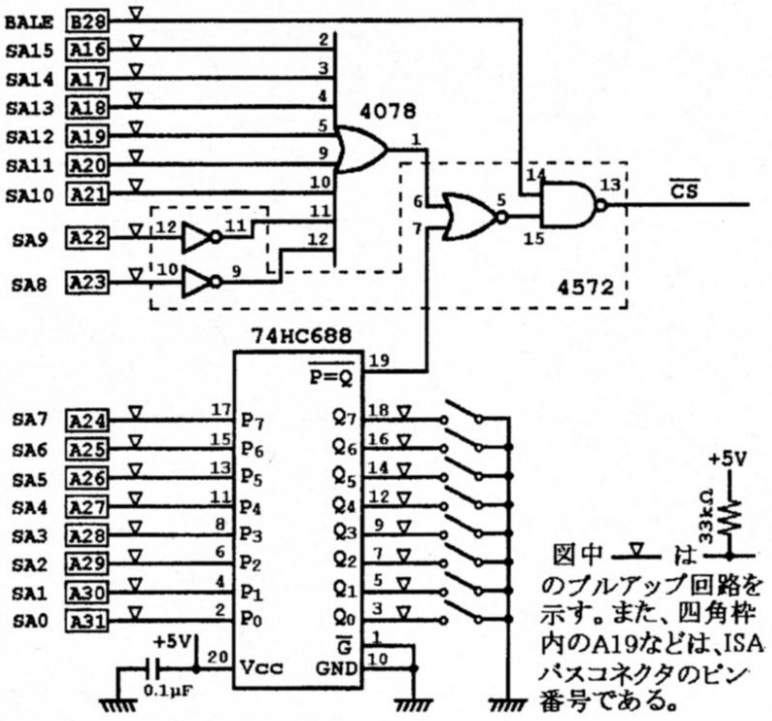


図 2: チップセレクト信号生成回路の例

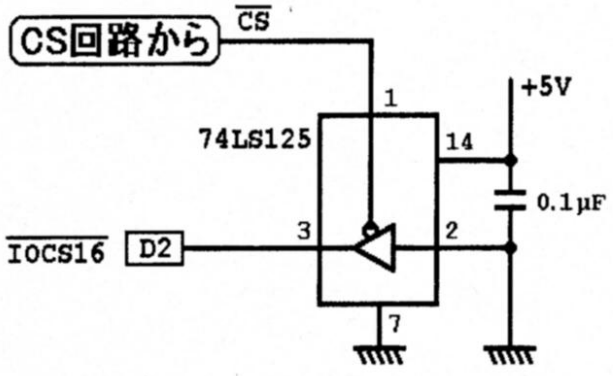


図 3: IOCS16 信号生成回路の例

4.3 温度計回路

この「ISA バスに差す温度計」の要になるのがこの部分です。ここでは、温度検出部に National Semiconductor 社製の IC、LM35DZ を、A/D コンバータとして MAXIM

社製の MAX150 を用います。

LM35DZ は、使用温度範囲 0〜80°C で、1°C 当たり 10mV を出力する、温度検出専用の IC です。この出力を、オペアンプの TL061CP(Texas Instruments 社製) によって増幅し、A/D コンバータに通して量子化します。A/D コンバータの MAX150 は、サンプル・ホールド回路と基準電圧回路を内蔵した 8 ビットのコンバータで、回路が簡単になる上に外付け部品も少なくて済み、グッドです。これらを使って、1/2°C 単位で 127°C まで測定できるようにしてみます。

まず、温度計のうち、温度検出部分の回路図を図 4 のようになります。ここでは、A/D コンバータに入力するまでを行います。もし、LM35DZ の出力を A/D コンバータの入力に直接与えたいのであれば、2kΩ の半固定抵抗器を外してしまって下さい。

ちなみに、この半固定抵抗器は、TL061CP によるアンプの増幅率を調節するために使います。

というのも、LM35DZ の出力は 10mV/°C ですが、A/D コンバータでは出力されるデジタル値を 1 だけ変えるには、(入力電圧範囲の上限/$2^{分解能}$)V/°C だけ入力が変わらなければなりません。例えば、入力電圧範囲が 0〜5V、分解能 8 ビットであれば、5/256 = 0.01953125 ですから、温度が 1°C 上がるときに A/D コンバータへの入力電圧が約 20mV 高くならなければならないことになります。この場合、増幅率を 2 倍にしておけば、1°C の温度変化で出力されるデジタル値を 1 だけ増やすことが出来るようになります。

増幅率を変えれば、12 ビットの A/D コンバータを使う場合や入力電圧範囲が 0〜2.5V 程度の A/D コンバータを使う場合にもそのままこの温度検出部分が使えます。

さて、A/D コンバータ部分の回路図はどうなるでしょう。図 5 を見てください。

MAX150 は、電源電圧 5V、アナログ入力電圧範囲 0〜5V で動作します。基準電圧は外付け・内蔵のどちらの回路も使えますが、この回路では内蔵の基準電圧回路を使用することにします。また、MODE 端子のレベルで、WR/RD モード (MODE=H) と RD モード (MODE=L) が選べますが、ここでは WR/RD モードを用います。これは、$\overline{WR}$ 信号の立ち下がりで A/D 変換を開始するモードです。RD モードについてはここでは述べませんので、興味のある方は MAXIM 社の HP(URL: http://www.maxim.com/) にデータシートがありますので、そちらをご覧ください (ただし英語)。

5 カードの使い方

このカードを利用するには、次のようなプログラムを書けば事足りるでしょう。ここでは、I/O ポートアドレスは 0x320H と設定したと仮定します。なお、このソースは FreeBSD 用です (まあ、他の PC–UNIX でもちょっと改変すればすぐ使えるでしょう)。

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

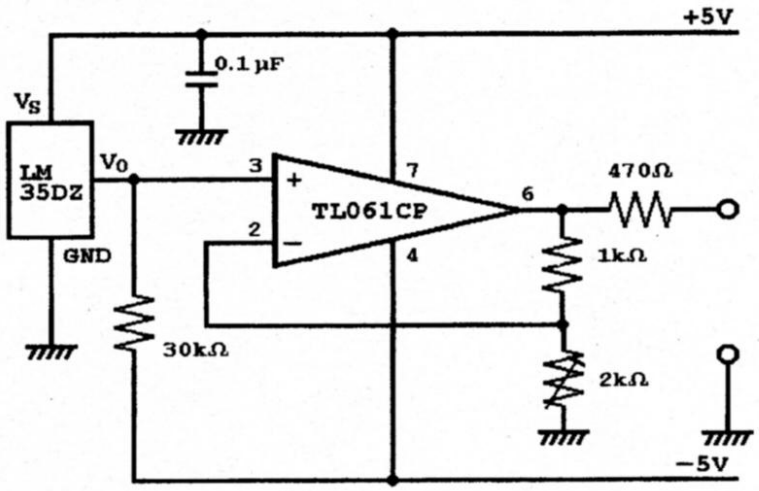


図 4: 温度計の回路 (温度検出部分)

```
#include <fcntl.h>
#include <machine/cpufunc.h>

#define DEVIO "/dev/io"
#define PORT 0x320

read_adc()
{
  int data;
  outb(PORT,0); /* ConvertStart*/
  while((data=inb(PORT))&0x8000!=0);
  return data;
}

main()
{
  int hndl,temprature;

  if((hndl=open(DEVIO,O_RDWR))==-1) {
    fprintf(stdout,"%s cannot open.\n",DEVIO);
    exit();
```

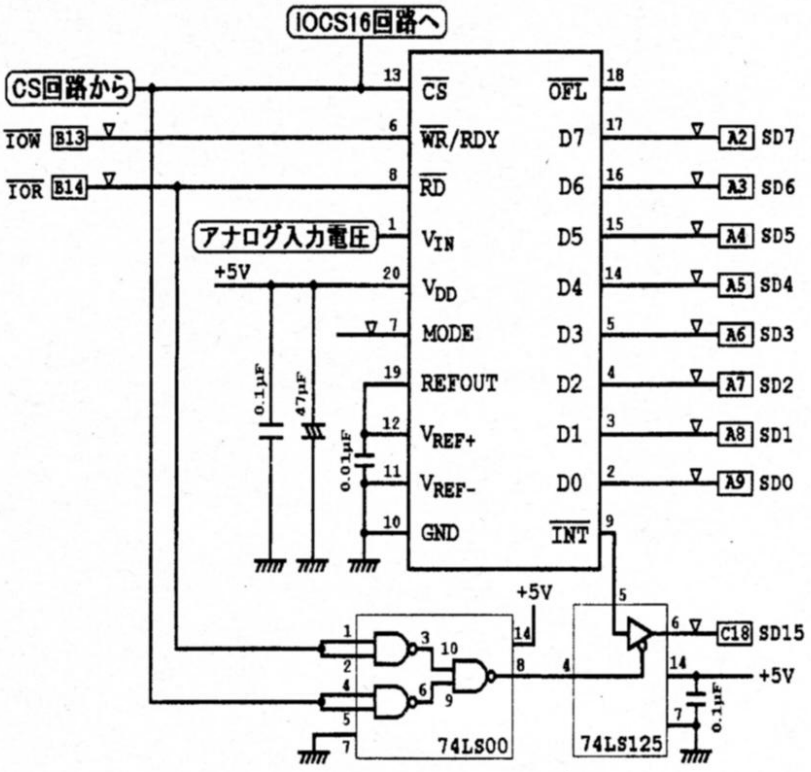


図 5: 温度計の回路 (A/D コンバータ部分)

```
  }
  temprature = read_adc();
  fprintf(stdout,"Now Temprature is %d.\n",temprature);
  close(hndl);
  return 0;
}
```

6 部品調達・製作・調整

ISA カード基板は日本橋のシリコンハウス共立などで 4000 円ほどで手に入ります。A/D コンバータの MAX150 は、どこへ行っても取り寄せのようで、入手が効きません。実際、私もこれで困りました。秋葉原のパーツ店の通信販売を利用するのがよいでしょう。

また、LM35DZ は、日本橋で購入すると 680 円ですが、秋葉原の秋月電子通商の

通信販売では250円ほどで買えますから、MAX150など他のパーツとまとめて買うといいかも知れません。パーツの値段はほとんど時価のようなもので、なおかつ店によってかなり価格が違うので、こういう事もままあります。

製作・調整については、製作しきれなかったので、多くは語れませんが、オペアンプの出力につながっている半固定抵抗器について少々。テスタと器、信用できる温度計を用意します(テスタはアナログよりはデジタル表示の物の方がお勧め)。器にお湯と水を入れて、きりのいい温度にします。50°Cぐらいが適当でしょうか。そこにLM35DZを浸けて(当然リード線が水に浸からないように)、アンプの出力電圧をテスタで計ります。半固定抵抗器を廻して、$T\times10\times(V_{Omax}\times1000\div2^R)=10000TV_{Omax}\div2^RmV$程度の電圧が出るように調整します(ここで、T: 温度, V_{Omax}: 入力電圧の最大値, R: A/Dコンバータの分解能(ビット数))。$T=50°C$、$V_{Omax}=5V$、$R=8bits$とすると、976.5625mV程度になる勘定です。厳密に正確な温度を求めるのでなければ、950〜1000mV程度になるようにすればよいでしょう。

あとがき

かなり悔いが残るものになってしまった。カードが完成できんではこの記事も裏付け無しのパーな記事じゃないか。細部の設計をもう一回やり直し、もっとグレードアップしたものを、ドライバ付きで作ることを、次回までの課題としておこう。IRQを使ったり、ステータスレジスタなんかも付けてみたいものだ。

教訓: 「貧乏暇なし」という格言は真理である。

簡単・便利なHTMLタグ集

機械システム工学科 2回生 仲田 慎平

☆☆☆☆☆　<HTML></HTML>

これが無いと、どうしようもありません。すべてのHTMLタグは、これに挟まれていなければ効果を発揮しません。

☆☆☆☆　<FONT></FONT>

文章を修飾するときなどに用います。このタグにはオプションがあって、

```
<FONT SIZE=5 COLOR="#000000">文字</FONT>
```

のように記述します。ここで"SIZE"というオプションは文字の大きさを１～８で決めます。また"COLOR"というオプションは文字の色を" "で囲まれた部分に、"\#"のあと赤・緑・青の順に光の強さを16進数で00～FFまでで書き込みます。数字の代わりに色を英語で書いてもOKです。これらはオプションなので、どれかが無くても大きな問題はありません。

☆☆☆☆　<BODY></BODY>

１つのHTML文書に対して１つだけ存在できるタグです。このタグにもオプションが存在しています。

```
<BODY BGCOLOR="#ffffff" TEXT="#000000" LINK="#ff00ff">
.....
</BODY>
```

というように記述するのですが、"BGCOLOR"は背景の色を決めます。"TEXT"で<BODY>タグに囲まれた部分の文字の基本色を決めます。"LINK"でまだアクセスしていないリンクを示す文字や枠の色を決めます。この他にも"ALINK"：リンクを示す文字もしくは枠をクリックしている最中の色。"VLINK"：アクセス済みのリンクを示す色。これらの色の決めかたは<FONT>タグの"COLOR"の決めかたと同じです。また、特殊なものとして、"BACKGROUND"：背景の画像を決めます。記述方法は、<BODY BACKGROUND="source">になります。" "で囲まれた部分（ここではsource）には、画像の場所と名前を記入します。

☆☆☆☆ <A HREF></A>

HTML 文書の最大の特徴ともいえる、"リンク"を示すタグです。これを用いた例を示すと

```
<A HREF="http://www.kit.ac.jp/">京都工芸繊維大学</A>
```

のようになります。ここでの"http://～"という部分は URL と呼ばれる、コンピュータネットワークにおける住所の様なものです。画面上に表示された、このタグで囲まれた部分がクリックされると、記述された URL を見ることができるのです。

☆☆☆ <HEAD></HEAD>

<HTML>のタグの中、<BODY>よりも上にある「ヘッダ」と呼ばれる部分です。文字通り、HTML 文書を人間の体にたとえるならば、頭の部分が<HEAD>に、からだの部分が<BODY>になる訳です。このタグに囲まれた部分にいろいろな特殊なタグを書き込むのです。このタグには特にオプションはありません。

☆☆☆ <TITLE></TITLE>

<HEAD>タグの中に存在する、その HTML 文書のタイトルを示すタグです。このタグに囲まれた部分の文章がブラウザ (ネットスケープ・ナビゲーターや、インターネット・エクスプローラーなどの、インターネット上の文書を表示するアプリケーション) のウインドウの左上に表示されます。また、しおり (ブックマーク) に記録しておくときには、このタイトルが見出しになることが多いです。

☆☆☆ <IMG>

画像などを表示するためのタグです。「イメージ」と読みます。オプションの中で最も重要なものが"SRC"です。

```
<IMG SRC="gazou.jpg">
```

のように記述します。" "で囲まれた部分に画像のファイルの名前を記述します。他のオプションの中で使いやすいものは、"ALIGN"です。これは、画像と文字を画面のどの辺りに表示するかを指定するものです。文字の、画像に対する高さを決める記述は、"top"、"middle"もしくは"center"、そして"bottom"です。文字の、画像に対する左右の位置を決めるものが、"right"または"left"です。例としては次の通りです。

```
<IMG SRC="gazou.jpg" ALIGN="left">……
```

このようにすると、……の部分の文字の左側に画像が表示されるのです。

☆☆　<CENTER></CENTER>

このタグに囲まれた部分は画面全体の中央に表示されます。ただし、ここで言う中央とは左右の空白のバランスがとれている、ということです。画面に表示しきれないほど横に長い場合は、このタグを使っていない場合と変わりません。

☆☆　<BLINK></BLINK>

これはリンクを表わすタグではありません。点滅 (**blink**) を示すタグです。その名前の通り、このタグで囲まれた部分の文字が点滅します。残念ながら、画像は点滅しません。また、このタグにはいまのところオプションは存在しません。

☆☆　<TABLE></TABLE>

表組みを作成するときに使用するタグです。主なオプションは"**border**"です。これは表の枠線の太さを決めます。

```
<TABLE border=2>……</TABLE>
```

のように記述するのですが、"**border**"という文字のみを記述したとき、自動的に太さは **1** にされるようです。"**border**"を記述しなかったときは、太さは **0** になります。

☆　<TR></TR>

<**TABLE**>タグの中で、行要素を示すタグです。<**TABLE**>タグの中にはこのタグがいくつか並んでいることが多いです。

☆　<TD></TD>

<**TABLE**>タグの要素を示すタグです。<**TR**>タグに挟まれて使われることが多いです。

```
<TR><TD BGCOLOR="#000000">……</TD></TR>
```

のように記述されます。"BGCOLOR"は要素の背景色を示します。指摘が無いときは<BODY>タグ内で指定した色が使われます。<BODY>でも指摘していなかったときは、ユーザの設定に依存します。他の主なオプションは"COLSPAN"・"LOWSPAN"そして"NOWRAP"です。"COLSPAN"は他の要素に対する横の要素数を、"LOWSPAN"は他の要素に対する縦の要素数を決めます。"NOWRAP"と記述すると、<TD>タグ内では改行をしないことを強制します。これらは、

```
<TD COLSPAN=2 LOWSPAN=3 NOWRAP>……</TD>
```

のように記述します。

☆ <HR>

横線を引きます。オプションが無いときは画面を水平に端から端まで線が引かれます。そのオプションですが、"WIDTH"や"SIZE"などがあります。"WIDTH"は画面の中でどの程度の長さの線を引くかを、ピクセル単位、もしくは、全体の長さに対する%で示します。"SIZE"は、線の太さを決定します。このオプションがないときは、サイズは2になっているようです。

おわりに

これだけ知っていれば、凝ってないHTML文書なら、サクサクかけると思います。楽しいホームページ運営ライフをお送り下さい。

L2 キャッシュの効能

岐津三泰

L2 キャッシュとは？

ここ一年ほど、PC の雑誌などをぱらぱらとめくると、

「L2 キャッシュ(level 2 cache)」

という文字をときどき見かけます。これは、「セカンドキャッシュ(second cache)」、「外部キャッシュ(external cache)」などともいわれ、よくアクセスすると思われるメインメモリの一部のコピーがおかれるメモリで、それを用いることによりメモリアクセスの高速化を図ろうというものです。これは、内部キャッシュ[1](internal cache)よりは低速でメインメモリより高速です。二、三年ほど前はこれがあるのとないのとでは体感的に違っていました。現在でもベンチマークテスト[2]では、

かなりの効果がある

とされています。たとえば iCOMP[3]値が、同じ動作クロックの Celeron と Pentium Ⅱ では後者のほうが高いことが挙げられます。しかし、最近の高速な CPU ではその効果が感じられなくなっている気がして、検証を行ってみました。

検証に使用した機材

検証には以下の二台のマシンを使用しました。これからはそれぞれのマシンをマシン名で呼ぶことにします。ここの用語に関しては逐一注釈を入れませんが、特に差し支えはないでしょう。もし知りたければ最寄の PC を組み立てたことがある人にでも聞きましょう。Guchio はコンピュータ部のマシン、NukuNuku は私所有のマシンです。ちなみに NukuNuku といっても

決して猫脳ではない

のでご安心を。

---

[1] CPU(中央演算装置; Central Processing Unit)内部にある超高速なメモリ。L1 キャッシュとも呼ばれます。用途は L2 キャッシュとほぼ同じ。たとえば Intel 社のものを挙げると、Pentium プロセッサでは 32kB、MMX Pentium プロセッサ、Pentium Ⅱ プロセッサでは 64kB。

[2] ある一定の計算などを行い、対象に負荷をかけて、その性能を計測しようとするもの。

[3] インテルが自社の CPU を評価するためのベンチマークテスト。

マシン名: Guchio
CPU: Intel Pentium II プロセッサ 266MHz (L2 キャッシュ 512kB 内蔵)
マザーボード: A-TREND ATC6220
ノースチップ: Intel 440BX
サウスチップ: Intel PIIX4
BIOS 製造元: AWERD
メモリ: SDRAM(CL=2) 64MB
ハードディスク: Wenstern Digital Caviar21220
インタフェース: E-IDE
容量: 1.2GB
グラフィックカード: ASUS AGP-V3000
チップセット: nVIDIA RIVA128
バス: AGP X1
ビデオメモリ: 4MB SDRAM

マシン名: NukuNuku
CPU: Intel MMX Pentium プロセッサ[1] 200MHz
L2 キャッシュ: バースト SRAM 256kB(パリティ有り)
マザーボード: NEC G8-VAZ
ノースチップ: VLSI Wildcat
サウスチップ: NEC STAR ALPHA 2
BIOS 製造元: NEC
メモリ: パリティージェネレータ搭載ファストページ DRAM(70ns) 8MB x 4
ハードディスク: IBM UltraStar 9ES(DCAS-34330U)
インタフェース: Ultra narrow SCSI
容量: 4.3GB
SCSI カード: IO-DATA SC-UPCI
チップセット: Symbios 53C845
グラフィックカード: Creative Graphics BLASTER Exxtreme
チップセット: 3Dlabs PERMEDIA2
バス: 32bit PCI
ビデオメモリ: 4MB SDRAM

検証の方法

検証には実際に日ごろ待たされるような状況を用いるよう工夫しました。これにより、この検証が

より現実に即したもの

であることを期待しています。というのもベンチマークテストは必ずしも普段

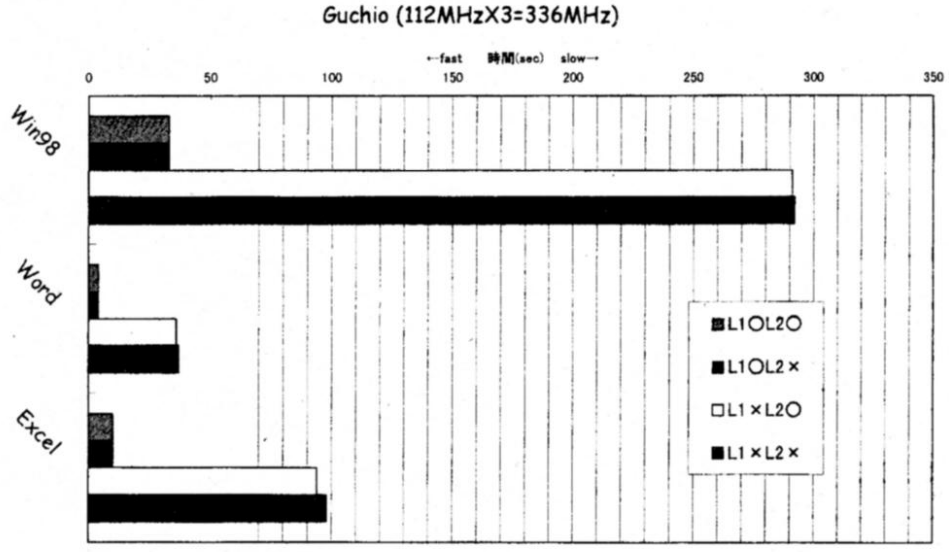
---

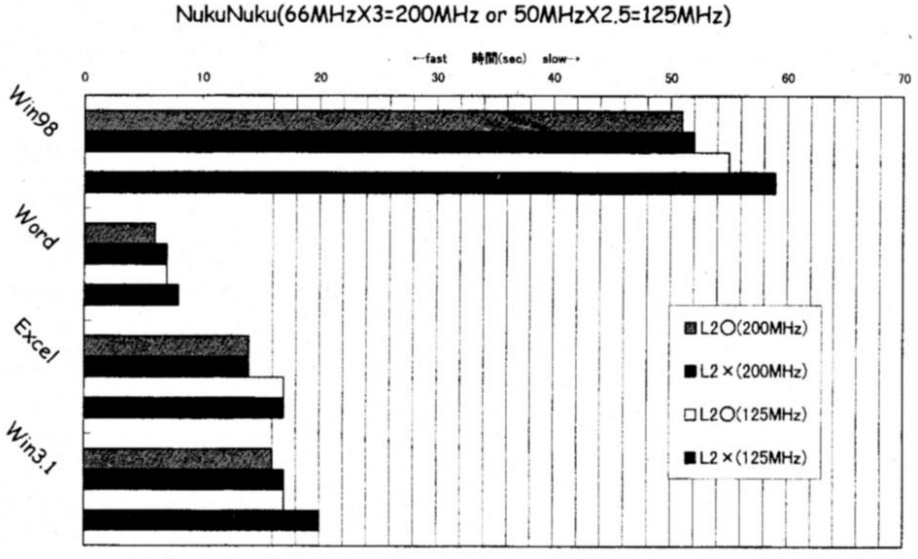
[1] メーカ不明の台湾製電圧降下ソケット「Multimedia CPU Upgrade Kit P55C-K6-L」を使用しています。日本での販売代理店は超級電脳。

の作業と一致するところを測定するとは限らないからです。

具体的には、Windows98 の起動時間、Word97 の起動時間、約 300kB の Word 文書に約 900kB の Excel の表を貼り付けたものの表をダブルクリックして編集可能状態になるまでの時間、そして NukuNuku のみ MS-DOS6.2 から Windows3.1 の起動時間を測定しました。

測定結果





考察

ミドルシップモデルと考えられる Guchio では、L2 キャッシュの効果はほとんど得られていないことがわかります。これでは L2 キャッシュの効能は

温泉の効能[1]並に怪しい

といわざるをえません。また、今回の趣旨とは外れますが L1 キャッシュを切っての測定も行ってみました。L2 キャッシュと比べればその効果は相当なものであることがわかります。こんな風に PentiumⅡのメモリアクセス遅さを実感するとは思いませんでした。

少し前のエントリーモデルと考えられる NukuNuku(200MHz)では、わずかに効果があります。さらに性能の低い NukuNuku(125MHz)ではその効果がそこそこ現れています。つまり、CPU の速度が上がれば上がるほど Windows98 では L2 キャッシュの効果がないということになります。これは、数 MB にも及ぶ実行ファイルが大量のファイルをリンクして実行する Windows98/95 では、

ミスヒット[2]が多発する

ためだと思われます。それは、NukuNuku(200MHz)でも Windows3.1 ではそこそこの効果があることからも予想されます。それでは、なぜベンチマークテストではよい結果を出すのかという問題は、ベンチマークソフトのサイズの小ささが挙げられます。つまり、プログラムがすべてキャッシュに乗ってしまうような大きさの、要するにメインループが 500kB 以下であるようなベンチマークソフトは理想的な状態を計測しているにすぎないのです。たとえば最初に挙げた iCOMP や、Super-PI といったものがこの種のベンチマークの典型例といえます。逆に、さらに小さい HDBENCH や WINSOCK などはメインループが 60kB 以下のため L1 キャッシュにすべて乗ってしまい、L2 キャッシュの有無や容量、性能に全く左右されなくなります。もっとも、最近のメガバイト級のベンチマークソフト（Intel Media Benchmark や 3DWINBENCH など）はある程度信頼できる値を出してくれるようです。

では、コードサイズの小さいメモ帳やワードパッドでは L2 キャッシュの恩恵が受けられるかというと、日本ではそれも難しくなります。なぜなら、その裏でメガバイト級の IME が交互に動作します。そのため

せっかく L2 キャッシュに乗っていたものが
クリアされてしまう

ため、結局はほとんど効果が得られないことになってしまうのです。Intel PentiumⅡはメモリアクセスが早くないので、その性能をコア内部のクロック

---

[1] 一応、お役所が調べているらしいですが。

[2] 必要なデータやプログラムがキャッシュメモリ上にないこと

の半分で動作する L2 キャッシュで補っているという感がありますが、その容量 512kB というのはもしかしたら米国向けなのかもしれません。少なくとも、プログラムが 1MB 超、辞書が 5MB 超(IME95 の例で、ほかはもっと大きい)というような IME を常に動かしている日本では

2MB〜4MB ぐらいの容量が必要

といえるのではないでしょうか。512kB という L2 キャッシュは今の標準的ユーザの使用形態[1]から考えますと、はっきりって、

グリコのおまけより無意味

なのかもしれません。なにせおまけなしの Celeron やメモリアクセスの早い AMD K6 シリーズ[2]（おまけはマザーボードに内蔵☺）のほうがはるかに安いのですから。

おわりに

今回の検証を通してさらに Intel が謎になってしまいました。このような事実を知っているのか、それとも知らないのかがです。iCOMP 値に固着して Pentium II の L2 キャッシュを 512kB[3]にしているのかと思えば、Celeron のような CPU を出しました。その後 Celeron に 128kB の L2 キャッシュを搭載したものがでました。この行動をどう思いますか？

1. 実は何も知らない
2. ユーザが知らないと思って儲け主義に走っている
3. 謎理論により

やはり 3 番ですかねぇ。

おまけ

じつは NukuNuku は、気持ち悪い AT 互換機ではなく、たちの悪い鎖国マシン PC-9800 シリーズなのでした。もとは PC-9821Xa7/C4 と呼ばれていたもの。今では PC-9821Xa200/C40 に…。

---

[1] ネットサーフィンやマルチメディアといった大きくて多数のデータを扱うようなことを毎日のようにやっていますよね？

[2] Pentium に互換を持った CPU で、近々K6-2/400(3D Now!という 3D 処理を高速化する命令セットを持った K6 の 400MHz 版)がでます。これは 6 倍設定をもっているそうな。

[3] L2 キャッシュが 256kB 以上になるとほとんど iCOMP 値が上がらず、512kB でほとんど頭打ちだそうです。

コンピュータは特殊な道具

機械　一回生　　外川内　竜行

コンピュータは特殊な道具である。と、いうよりも道具かどうかも怪しい所がある。コンピュータはどんどん進化している。しかし、それも正確な意味で進化しているのかどうかは判らない。なぜなら、コンピュータにはその道具としての目的、つまり、何をするための道具であるかと言う事が定義されていないからだ。

ねじ回しはもちろんネジを回すものである。その場所にあるネジを締めたりゆるめたりできればいいのである。では、コンピュータは何をする道具だろう。そんな物は定義されていない。当然である。コンピュータの歴史を少し勉強したら分かる事だが、コンピュータは今ある機能よりも更にすごい機能を持たせようとして発展してきているからである。つまり、今現在のコンピュータができる事なら述べられるかもしれないが（それも、相当大変だが）明日のコンピュータに何ができるかは分からないのである。しかし、昨日のコンピュータも今日のコンピュータも明日のコンピュータもコンピュータである。結局何ができる道具なのか分からないのである。

それでは、「何ができる」ではなく「何をする」道具なのだろう。その答はもちろん計算（演算）なのだが、それはこの問いに関する正しい答えではない。なぜなら、「ねじ回しとはモーメントを云々する道具です。」というのは正しい答えではなく「ねじ回しとはモーメントを云々してネジを締める道具です。」または、「ネジを締める道具です。」と言うのが正しい答だからだ。では、今回の質問の意味での正しい答は何だろうか。分からないのである。それは、明日のコンピュータの事が分からないのと同じように、明日の技術者がコンピュータに何をさせたいかが分からないからである。

コンピュータとはまったく変な道具である。「何ができるか」も「何をするためのものか」も定義できないのである。（今回の記事を書くに当たって本を探してみたが広い意味でこれを定義してるものはなかったし、そういう事を書いてある本もほとんど見つからなかった。）つまり、モーメントとは「何ができるか」「何をするためのものか」を考えるのと同じようなものである。コンピュータは確かに存在している道具としては全く異色の存在と言えるのである。

しかし、それが結論では記事にならないのでここでもう一つの事を考えてみようと思う。それは「コンピュータの最終形とは何か」である。これは簡単である。なぜなら、明日のコンピュータの事は分からないとは書いたが、明日のコンピュータの方が今日のコンピュータより優れているのは当たり前だからである。それと同じくして、明日の技術者がコンピュータにさせたい事はコンピュータが今までできなかった事であるからだ。つまり、誰かが技術としてコンピュータを発展させようとする限り、コンピュータは計算（演算）をもとにして知的資源（情報）を扱う事なら何でもできるようになろうとする方向に向かう事になる。自分にはコンピュータがなんでもできるようになるかは分からないし、今と同じしくみのまま進歩して行くかもわからない。しかし、コンピュータが人間の感情を理解できるようになったり、本当の意味で命令を与える事なく自立的に判断ができるようになれば、と多くの人が夢見るはずである。つまり、人間自体がなんでもできるようになろうと科学と技術を進歩させていく中で、その情報の部門を受け持つのがコンピュータなのである。（だから、道具の形としては定義しにくい）そして、今存在しているものはその未完成な形と言う事になる。
人間がなんでもできるようになるかは別の議論としても、コンピュータ技術の未来は

おおむね明るい。（今のところそれを否定する話は聞かない）コンピュータが産み出される前までは情報を扱う道具は「情報を表現する」「情報を広める」（前者はキャンパスと絵の具、後者は新聞やテレビのなどの事である）の二つの機能しかなかった。そして、コンピュータにはそれ以外のさまざまな機能がある。しかし、前に書いたように未完成品であると言う要素も含んでいる。だからこそ、これからは見える形にしろ見えにくい形にしても人間がコンピュータに頼るようになる事は増えていくだろう。コンピュータが自分が生きている間にどこまで完成に近づくかも、自分がその世界のプロになるかも分からないが（自分は機械科の学生である）死ぬまで付き合う事になるだろう。コンピュータ部に所属しているからには興味を持ってその行程が少しでも理解できるようになりたいものだ。

編集後記

今年も例によって期限ギリギリまでの作業となった。日頃怠惰に過ごしていると、こんなときにえらい目に遭うものだ。反省が必要である。「反省だけなら、サルでも出来る」から、反省の上に立って精進を積まねばなるまい。

さて、今回は LaTeX を用いた原稿と MS-Word を用いた原稿とが混在し、妙な趣を醸し出しているが、いかがだろうか。念のために言っておくと、決して編集者が手抜きをしたわけではない。個性的な原稿をそのまま読者に伝えるための便法だと思って頂きたい。

それはともかく、今年は例年と異なり、ロボット製作に乗り出すなど、新機軸が打ち出された年だったように思える。チームを組んでの製作は、知識の補完なども含めて、非常に有益だということを改めて実感した。それとともに、各人各様で自らの力量をチェックするいい機会になったのではなかろうか。

それでは皆様、少し早いですが、よい御年をお迎え下さい。

編集人：　服部　保
発　行：　京都工芸繊維大学コンピュータ部

1998 年 11 月 21 日 初版発行

らいむでチュウ