Linux Japan 11月号への掲載原稿です.
26char/46line
#10203040506070809101112131415161718192021222324252627282930

「Linux/Alphaによる数値計算ならびにRISCアーキテクチャのおはなし(第五回)」
-- SRMでのLinux,NetBSDインストールとAlphaのアセンブリ言語のおはなし
清水尚彦 nshimizu@et.u-tokai.ac.jp

1. はじめに

IntelのMercedの出荷が2000年に遅れることになってワークステーションメーカーはかなり困っているようですね. この遅れの結果2000年までの間はAlphaチップの性能優位は揺るがないものになりそうです. その主力チップである21264のマシンも実際に動作をはじめたようでSPECの値も報告されています. AlphaServer 8400 6/575では期待通りSPECint95は30を越えています. SPECfp95の値は47.7とはじめのアナウンスより幾分低い値ですがまだ最終的なチューニングは終っていないと思われるので数値自体は更に伸びると思います. また, 8 CPUでのSPECfp95は114ということです. もっとも最初のアナウンスは500MHzのマシンのSPEC値がこの程度という話だったので話が違うという人が多いのですが, この雑誌が出るころには違う値が報告されていると思います.

9月号でAS200を購入したと書きました. さっそく到着して確認したところやっぱりBIOSにはSRMしか入っていませんでした. ROMのソケットはありますのでおそらくROMを買って来てBIOSを追加することはできると思います. とにかくSRMから立
ち上がるようにインストールを試みました. せっかく買うマシンだから一つのOSだけでは面白くありません. そこで, 最近インストールがたいへん簡単になっているNetBSDを合わせて導入することにしました. その他にもDebianとかRedHatとか試してみたいOSはいろいろありますがこれらは外付けのHDDを購入したときのお楽しみとしてとりあえずは二つのOSの導入を計ります.
購入したのはDECのAlphaStation 200 4/100とS3のビデオカードです. インストールには別のLinuxマシンが一台あった方が便利です. (というか単体でのインストールはやったことがありません. CDROMをつければ比較的簡単かもしれません) ついでに書きますと, この4/100というマシンは21064というチップを使っているとカタログではなっていますがやって来たマシンは21064Aという次の世代のプロセッサが搭載されていました. おそらく古いチップはすでに作られていないのでしょうね. クロックは水晶発振器から供給されているので簡単にはクロックアップはできないのですが, マザーボードの説明書ではAS200では使わないとされているPLLのジャンパスイッチが載っているのでこのあたりをいじるともしかしたらクロックアップができるのかもしれません. 水晶だけでクロックアップしようと思うと400MHzとかの水晶が必要になるのでなかなか入手は難しそうです. CPUは周波数も低いのでファンなしの放熱器を使っています.

つぎに久々にAlphaのアセンブラの話を再開します. RISCとはいってもAlphaには数多くの命令があります. この命令をいちいち説明していたら誌面がいくらあっても足りないので後の説明に使う程度のサブセットだけを説明し詳細は各種資料に譲ります. では何を書くかというとアプリケーションインターフェイス(API)のことになります. Alphaに限らず多くのプロセッサでは関数や手続きの呼出方法に関連したレジスタやメモリの使い方がAPIとして定義されています. このAPIを知らないと自分でアセンブラで独自の関数を作る時に困ってしまいます. (関数を作ったのは良いけれど高級言語から呼出ができないと使い勝手が悪くてかないませんね)
PattersonとHenneseyの有名な教科書「Computer Organization & Design」は邦訳が出ている第一版ではAPIの説明なしにAPIに沿ったアセンブラの説明をしていますが, 説明に無理があったのか第二版ではさすがにAPIの説明を導入しています. 彼らの本はMIPSアーキテクチャに沿って説明していますが第二版を入手できる方は考え方の参考になるので入手してみて下さい.

2. SRMコンソールを使ったLinuxとNetBSDインストール

Alphaのマザーボードは3種類のBIOSが載っている可能性があります. 一つはDigital UNIXやOpenVMSに使われるSRM, あとの二つはどちらもWindowsNTに使われるARCとAlphaBIOSです. 最近のマシンではARCではなくAlphaBIOSが使われることが多いようですが, どちらもあまり変わりません.

Linuxはどの BIOS でも立ち上がるのですが，実は NetBSD は SRM でしか動作しません．そこで，今回のように両方ともインストールしたいという場合には SRM しか選択肢はありません．ここで，若干困ったことがあります．というのは SRM ではブートできる OS は一つのデバイスに一つだけです．これはブートの仕組みが非常に単純なことに起因しますが，二つの OS を切替えて使用したい向きには困った仕様です．LILO のようなブート切替えのプログラムを書けば済むはなしですが x86 と異なり Alpha の Linux では NT 以外にはデュアルブートの必要を感じていないらしくそのようなプログラムはありません．aboot というプログラムが近いことをやってくれますが，Linux を対象としているので NetBSD とのマルチブートには別の手段が必要とします．

さて，そこでここではLinuxをフロッピーからブートし，NetBSD をハードディスクからブートするような仕組みを作ることにします．
カーネルをフロッピーからブートするのは時間がかかりますが，今回は MILO だけをフロッピーにいれてカーネルは HDD にいれておくことにします．
MILO のほうが容量が小さいのでこれによってブートにかかる時間が節約できます．

さて，私の購入したマシンにはたまたま SRM しか入っていませんでしたが，ARC もしくは AlphaBIOS のマシンを SRM に入れ換える場合には DEC の WEB からファームウェアをダウンロードして来る必要があります．
入れ換えの方法はマザーボードの説明書等を参照下さい．

SRM はシリアルポートからコントロールできるのでこの機能を使うと本来はビデオカードが無くても使えそうです．ところが，NetBSD はその通り使えるのですが Linux ではインストールの途中でキーボードを直接見に行くプログラムがあってシリアルポートからだとそれ以上インストールが進まないという自体に陥ります．ラムディスクドライバにパッチを当てれば簡単になおるとは思いますが一般的ではないのでここではビデオカード経由のインストールをします．

2.0 NetBSD のインストール

NetBSD-1.3.2 は Linux と異なりインストール用のフロッピーは一種類しかありません．そこで，あまり迷わずにインストールできるのですが，インストーラは install を選択すると既存のパーティションを無視して上書きしてしまうので Linux との共存のために先に NetBSD を入れておきましょう．

NetBSD-1.3.2 はとりあえず次のファイルをフロッピーに書き込んでインストール用のディスクを作成します．

install132.fs

install132.fs のフロッピーを入れて SRM コンソールから

boot dva0

と入力してブートします．
こちらは SRM からのブートを前提にしているので非常に簡単にインストールできます．
ただし，NetBSD ではディスクのジオメトリデータを要求するのであらかじめシリンダ数，セクタ数などの情報を入手しておかなくてはなりません．
インストールかアップグレードかの質問に

install

と答えるとディスクパーティションの設定を含むインストールのプロセスが動きます．

2.1 ディスクパーティション

Linux だけで使う場合には FDISK タイプのパーティションを使うことができますが今回は NetBSD で disklabel 形式のパーティションを作ります．
Linux にも disklabel 形式のパーティションを操作する minlabel というコマンドがありますが NetBSD の disklabel と必ずしも互換性が保障されないので使わないでください．

disklabel 形式では通常パーティションの b と c は特別の目的に確保され

ます．そこで，次のように分割することにします．AS200 には 2GB のドライブが内蔵されているのでこれを適宜分配します．
容量の配分は多分に趣味の問題が絡んでいますのでこれに拘る必要は全然ありません．

区分	説明	容量	形式
1	NetBSD ルートファイルシステム	64MB	ffs
2	NetBSD/Linux スワップ領域	64MB	swap
3	ディスク全体		
4	Linux(全て単一のパーティションにします)	961MB	ffs
5	NetBSD /usr ファイルシステム	963MB	ffs

Linux の UFS は信頼性に欠けるところがあるので Linux から NetBSD 領域は読みだし専用としておきます．（本当はホームディレクトリを UFS で共用できると嬉しかったのですが x86 で試しにやってみたところ書き込みをするとかなりひどくファイルシステムがダメージを受けたので止めておいた方が無難です．）本来は/var を分けたりするのが一般的なのでしょうが，ホームディレクトリを NFS で x86 のものをマウントするつもりなのでクラッシュの被害はそれほど大きくないと思われ以上のようにします．

Linux のパーティションは ext2 ではないかと気がついた方もいると思います．実は Linux のインストーラを使いたくないのでとりあえずパーティションだけを先に確保しておきます．Linux では mke2fs でファイルシステムを構築すればパーティションテーブルに書かれた名前とは無関係に ext2 で利用できますので安心してください．

次に，FTP 経由で NetBSD のディストリビューションを入手するために

```
ifconfig lo0 127.0.0.1
```

ととりあえずローカルループバックの設定をします．
次に

```
ifconfig -l
```

とネットワークの認識を確認した後
10baseT のインターフェイスで接続する場合には例えば次のように入力します．

```
ifconfig de0 192.168.1.3 netmask 255.255.255.0 media 10baseT/UTP
route add default 192.168.1.0
```

次にディスクをマウントした後最低限
ftp によって binary のディレクトリにある
base.tgz, etc.tgz, kern.tgz を取って来ます．
ここまでの操作でハードディスクは/mnt の下にマウントされていますので
ここに必要なファイルを取って来ます．

```
cd /mnt
ftp 192.168.1.1
```

次にデバイスファイルを作るために次のコマンドを入力します．

```
cd /mnt/dev
sh ./MAKEDEV all
```

ここまでできればできたようなものです．
次のコマンドで一旦 SRM に戻りましょう．

```
shutdown -h now
```

次に SRM でハードディスクからのブートを指示します．

```
boot dka0
```

いかがですか? NetBSD がうまく立ち上がったでしょうか?
この段階ではまだ/etc/rc.conf の編集をしていないのでシングルユーザーでし

か立ち上がりませんが後の設定は人によって大きく異なるので詳細は示しませんが
大きな山は越えたので各自の設定を楽しんでください.

2.2 MILO と SRM

実は一般に入手できるMILOはSRMと相性が良くありません. 特にメモリサイズ
を正しく認識できないのは困ったものでした. しかしNikitaさんがMILOのパ
ッチを発表して事態は改善されています. 下記のサイトにパッチといくつかの
バイナリが収められています.

ftp://genie.ucd.ie/pub/alpha/milo

このMILOはSRMからメモリサイズを取得できるだけでなくブートのためのパラ
メータブロックを持つことができます. 通常ARCではブートのオプションなど
を記憶しておくことができますが, MILOからNVRAMが使えない場合には自動的
にブートする手段がなくなります. ところが, NikitaさんのMILOではディスク
の一部にブート時に実行するコマンドを記憶することができるのでARCやNVRAM
が使えなくても自動ブートできるようになるそうです.

さて, これで準備は整いましたか? ちょっと待って下さい. SRMからブートで
きるMILOが作られていません. 実は上記サイトにはARCからブートできるよう
に作られているMILOが入っていますが, SRMからブートするためには少し工夫
が必要です. この工夫のためにはLinux/Alphaがインストールされている必要
があります. ですからここでは一旦RedHatにあるavanti-s.imgというフ
ァイルを使ってLinuxのインストールを進めておきます.
Linuxがインストールできたらそのツールを使って後でSRMからブートするMILOを
作ります.
このブートのためのイメージファイルはabootというコマンドを使って
ブートするイメージファイルになっています. abootをインストールの主役に
する方法もあるのですが, いろいろな設定においてMILOの方が将来性がありそ
うなのでこの記事ではMILOを中心に書きます.

2.3 カーネルのブート

Linux/AlphaではマザーボードによってカーネルがDECが作っていないからですが, 逆に時代の進歩をどん
どん採り入れたマシンを作りやすい仕組みともいえます(?).
でもNetBSDではインストーラもカーネルも一つに統一されているのでユーザー
の便利さから言えば本当は一つにして欲しいところです.
この記事で
はAlphaStation 200 4/100（このシリーズのことをAS200もしくはAvantiと呼びます）
を中心に説明しますが, 他のマシンを使う方は必要
な項目を読み換えて参照下さい.
まず, StatabowareのルートファイルシステムとRedHatのavanti-s.imgを用意
してください.
Stataboware の執筆時点の最新版は1.4β です.

ftp://ftp.eni.co.jp/pub/mirrors/Linux-Alpha-JP/ftp.statabo.rim.or.jp/

に必要なファイルが格納されています.
binarykitというディレクトリにはLinuxの配布イメージが収められています.

RedHatから持って来た
avanti-s.imgをフロッピーに書き込みますが, x86のLinuxなどでは次のようにします.
(こんなことはLJの読者なら当り前ですよね.) 同じようにしてルートファイル
システムのフロッピーも作成しましょう.

```
cat avanti-s.img > /dev/fd0
```

SRMからブートするにはブートのコマンドを発行します.
このフロッピーをAS200のドライブに挿入し, SRMのプロンプトから

```
boot -file vmlinux.gz -flags "root=/dev/fd0 load_ramdisk=1" dva0
```

と打ち込むとカーネルの読み出しを行い, 途中でディスクの入れ換えを要求し
ます.

ディスクを入れ換えてエンターキーを打ってしばらくして

 login: メッセージが出れば成功です．これで半分は終
ったようなものです ?_?

次はあらかじめ作っておいたパーティションにファイルシステムの作
成を行います．私と同じように作った人は次のコマンドでLinuxのファイルシ
ステムを作ることができます．

```
mke2fs /dev/sda3
```

実はここのところで少し混乱するかも知れません．minlabelではすべてのパー
ティションに順番に番号をつけて表示しますがLinuxのデバイスドライバは有
効なパーティションだけを番号付けするようです．

2.4 Stataboware のインストール

前出のFTPサイトから最低限次のファイルを取って来ます．
残りのファイルは必要に応じて取って来てください．

```
root.tar.gz
usr.tar.gz
var.tar.gz
```

これに自分のマシンに対応したカーネルを用意して下さい．
私はとりあえずはRedHatのavanti.gzを持って来ました．RedHatにはSRM用の
avanti-s.gzもあるのですが，MILO経由でブートする場合にはこちらは使いま
せん．

次に先程ファイルシステムを作ったパーティションをマウントします．

```
mount /dev/sda3 /mnt
cd /mnt
```

マウントできたらこのディレクトリに用意したファイルを転送します．
AS200へのファイルの転送はx86のマシンからftpコマンドで行いましょう．
ネットワークを活性化するために次のコマンドを入れます．
(ネットワークアドレスは状況に応じて設定してください)
SRMのコンソールではネットワークインターフェイスの自動認識はしないので
すがLinuxでは自動認識しますので10Base-Tか10Base-2のどちらかに必要な配
線をするだけで準備はできます．

```
ifconfig eth0 192.168.1.2 netmask 255.255.255.0
route add -net 192.168.1.0 eth0
```

その後通常通りftpコマンドでローカルのx86マシンからファイルを転送しまし
ょう．(もしはじめてのコンピュータがこの機械だったら? うーむ，どうしま
しょう? 一応RedHatのカーネルにはPPPも組み込まれているので必要なファイルを
取って来ればPPPで接続できるはずですが..)

必要なファイルがすべて転送されたら次にファイルを展開します．

```
for i in *.tar.gz
do
 tar xzvpf $i
done
```

として入手したファイルを展開しましょう．
その後，Linuxのカーネルも同じディレクトリに転送しておきます．

必要なファイルがすべて転送されたら次にファイルを展開します．

```
for i in *.tar.gz
do
 tar xzvpf $i
done
```

として入手したファイルを展開しましょう．
その後，Linuxのカーネルも同じディレクトリに転送しておきます．

ここまで準備ができたらfstabを作成します.

```
cd /mnt/etc
cat > fstab
/dev/sda3       /           ext2        defaults   1   1
/dev/sda2       swap        swap        defaults   1   1
none            /proc       proc        defaults   1   1
^d
```

ここで^dはコントロールDのことです. コントロールキーを押しながらDのキーを押して下さい.
打ち間違ったらもう一度catコマンドを入力するところからやり直して下さい.

さて, ここまで来ればあと一歩です. インストールの確認をしてみましょう.
まずはせっかくインストールしたシステムをハードディスクに確実に書き込むためつぎのコマンドを入れて下さい.

```
sync
cd /
umount /mnt
```

これでディスクはマウントから外されました. それでは一旦リブートしましょう. もう一度avanti-s.imgのフロッピーにご登場願いインストールの確認をします.
リブートでSRMのコンソールプロンプトが出たところで

```
boot -file vmlinux.gz -flags "root=/dev/sda3"
```

と入力してブートします.

2.5 MILOフロッピーの作成

NikitaさんのMILOをSRMで動作できるようにするためのツールはカーネルをコンパイルする作業によって合わせて作成されます. そこで, MILOを作る前にまずはカーネルのコンパイルをしてしまいましょう.
Linuxのカーネルはいろいろなサイトにあるので適宜取って来てください.
MakefileのARCH変数をalphaに変更するのを忘れないように.

```
cd /usr/src/linux
make config
make dep
make boot
```

でカーネルのコンパイルができるのですが, make configのパラメータでいくつかの注意があります. まず, マザーボードは必ず自分のマシンにあったボードを選択して下さい. つぎにTGAを本当に必要とする人以外はTGAの組み込みをしません. その他のデバイスドライバは必要に応じて組み込んで下さい.
あと, SRMを使ってはいますがMILOからのブートを行うのでSRMの利用はNと答えて下さい. このオプションはブートローダを使わずに直接カーネルをディスクの先頭に書き込んでブートする場合に利用するものです.
これで作ったカーネルはブートのために使えるのですが, その前にカーネルを作った時に用意されるツールを利用してMILOを作っておきましょう.
前出のNikitaさんのFTPサイトから自分のマシンに合ったMILOを取ってきます.
私の場合には avanti.free を入手しました. これをホームディレクトリに置いておくものとします.
つぎのコマンドによってブート可能なMILOフロッピーを作ります.

```
/usr/src/linux/arch/alpha/boot/tools/objstrip -v -p avanti.free mboot
cat mboot avanti.free > /dev/fd0
```

以上のコマンドによってブート可能なフロッピーディスクが作成できます.
このフロッピーでブートしてみましょう.
無事MILOのプロンプトが出れば成功です.
それではMILOからLinuxをブートします.

```
boot sda4:vmlinux.gz root=/dev/sda3
```

MILO と Linux のパーティションの番号が
違っていますが，そういったものだと思ってください．
利用する MILO によってはこれが一致するものもあるようですが，
パーティションを MILO がどのように認識しているかは次のコマンドで
大体把握できます．

```
show device
```

このコマンドで認識しているパーティションの数を確認してください．
ディスクラベル形式ではドライブ全体をパーティションの一つとして登録して
いますがこれが Linux や MILO では無視したりしなかったりするようです．

いかがですか? 無事 Linux が立ち上がりましたか?
NetBSD と同様にこれからの設定は各自の好みで行ってください．

3. Alpha アセンブラ入門 (1) レジスタの使い方

今回から少し本格的に Alpha のアセンブラ入門編をはじめます．
初回はレジスタの使い方についてです．
次号まで待ち切れなくて英語に抵抗がない人は次の URL を参照ください．

http://www.unix.digital.com/faqs/publications/base_doc/DOCUMENTATION/HTML/AA-PS31D-TET1_html/TITLE.html

Alpha には整数と浮動少数点のレジスタがそれぞれ 32 本ずつあります．
レジスタの使い方はアプリケーションインターフェイスによって決まっていま
す．
それぞれの使い方は以下の通りです．

整数レジスタ

レジスタ名	説明	
$0	式の評価値	手続き呼出で保護されない
$1-8	一時変数	手続き呼出で保護されない
$9-14	局所変数	手続き呼出で保護される
$15	局所変数もしくはフレームポインタ(fp)	手続き呼出で保護される
$16-21	引数	手続き呼出で保護されない
$22-25	一時変数	手続き呼出で保護されない
$26	戻り番地	手続き呼出で保護される
$27	pv もしくは一時変数	手続き呼出で保護されない
$28	アセンブラ予約	手続き呼出で保護されない
$29	グローバルポインタ(gp)	手続き呼出で保護されない
$30	スタックポインタ(sp)	手続き呼出で保護される
$31	常に 0	

浮動少数点レジスタ

レジスタ名	説明	
$f0-f1	式の評価値(複素数の時に f1 を使う)	手続き呼出で保護されない
$f2-f9	局所変数	手続き呼出で保護される
$f10-f15	一時変数	手続き呼出で保護されない
$f16-f21	引数	手続き呼出で保護されない
$f22-f30	一時変数	手続き呼出で保護されない
$f31	常に 0	

手続き呼出で保護されないとしているレジスタはプログラム中で勝手に使って
構いません．逆に保護されるとしているレジスタは使う場合には例えばスタッ
ク中に一旦元の値を格納して手続きから戻る時には元の値に戻しておく必要が
あります．

ここで特徴的なレジスタとして gp があります．実はこの使い方は Alpha が 64 ビ
ットであることと深い関係があります．Alpha 以前のプロセッサではプログラ
ムが手続きの呼出をする時に 32 ビットのアドレスを生成するためにちょっとし
たトリックを使っていました．ところが 64 ビットの時代になるともはや小手先

のトリックではアドレスを生成するのは現実的ではなくなっています.
そこで, AlphaではGAT(グローバルアドレステーブル)という大域参照のアドレスを集めた表を用意することにしています. (これはメモリ中に置きます)
大域変数や手続き(こちらも大域的なアドレス変数とみなせます)は一旦この表から間接的に呼び出すことにして64ビットの定数の生成というトリックを不要にしています. ところが, そのために大域変数の参照には一旦GATからアドレスを読みだした後に間接参照する必要が出て来て性能上問題になります.
特に頻度が高い手続きや関数の呼び出しでは性能上もあまりうれしくありません.
これを解決するために特に同じソースファイル中に置かれている手続きではこの間接参照をスキップすることができるようになっています.
同一ファイルの場合にはコンパイラが処理できますが, リンクした結果プログラムの大きさがある程度小さい場合にはリンカが最適化する余地が残されています. 残念ながらLinuxのリンカはそういった最適化をしませんが, Digital Unixのコンパイラは最適化するようです.
このgpを設定するためにAlphaではldaとldahの2命令の命令が必要ですが, GATへのオフセットの計算値をこの2命令の変移に変換する作業は
結構大変なのでgpにGATの値をロードする命令として
アセンブラの疑似命令ldgpが定義されています.

では, 簡単なCの関数のアセンブラへの展開を見てみましょう.

```
extern int k;
int sample(int a)
{
 int j;
  j = foo(a);
  return j+k;
}
```

これをアセンブラに展開すると次のようになります.
まだ命令の紹介はしていないのですが, 展開のリストを理解するために簡単に紹介しておきます. ldaはロードアドレス命令でアドレスを計算した結果をレジスタに格納します. ldq,stqは64ビットのロード命令とストア命令, 同じくldl,stlは32ビットのロード/ストア命令です. addl,addqは32ビットと64ビットの加算命令, jsr,retはそれぞれサブルーチン呼び出し, サブルーチンリターンの命令です. 余談ですがAlphaではjmp,jsr,retは同じ命令でヒント情報が異なるだけです. ですから性能が低くて構わないならjmpと書くところをretと書いても構わないのです :-)

```
        .file  1 "sample.c"
        .version        "01.01"
        .set noat
gcc2_compiled.:
__gnu_compiled_c:
.text
        .align 3
        .globl sample
        .ent sample
sample:
        ldgp $29,0($27)
$sample..ng:
        lda $30,-16($30)
        .frame $30,16,$26,0
        stq $26,0($30)
        .mask 0x4000000,-16
        .prologue 1
        jsr $26,foo
        ldgp $29,0($26)
        lda $1,k
        ldl $1,0($1)
        addl $1,$0,$0
        ldq $26,0($30)
        addq $30,16,$30
        ret $31,($26),1
        .end sample
```

関数の最初のところでエントリが二つあります.

```
sample:
$sample..ng:
```

この二つは前に書いたgpをセットしない場合に備えたエントリになっています. リンク時もしくはコンパイル時にgpを設定する必要がなければ二つ目のエントリポイントに直接呼び出しがかかります.
gpの設定が必要な場合は次のケースのどれかに該当する場合です.

1) 他のルーチンを呼び出す
2) スタティック変数を参照する
3) 31ビットより大きな定数を使う
4) 浮動少数点定数を使う

jsrの呼び出しではレジスタ$27(pv)に
呼び出し先のアドレスを格納して呼び出すことになっているため
$27には関数の先頭アドレスが格納されています.
ldgpの引数で$27を使っているのは関数のアドレスから
GATのアドレスを計算するのです.
同じくサブルーチンコールの後に$26でldgpを使うのはリターンアドレスを元にGATの計算をすることを意味しています.

次回はこれらのレジスタとメモリの関係についてまとめて見ましょう.

4. おわりに

今回購入したAS200は
100MHzの21064Aという今ではたいていのPCにかなわないスペックのマシンですがなかなか頑張って働いてくれています.
あまり安かったので職場にも一台買ってしまいました. X端末として使ってもTeXによるDTP用としても不満を感じない程度の性能です. もっとも数値計算にはさすがに遅いとは思いますが..
Digital UNIXのライセンス付のマシンならDECのCコンパイラも使えるのでその方が嬉しい人も多いかも知れません.
UNIXライセンス付の同じ型のマシンも20万程度で入手できるようですのでDigital UNIXの資産が欲しい方にはちょうど良いターゲットかも知れません.

私は大学で「集積回路設計製作」という演習を教えているのですが, ここではLSI設計の基礎からハードウェア記述言語によるRISCプロセッサの設計まで演習します. 短い演習時間に参加者は良く頑張ってオリジナルのプロセッサを0から設計できています.
従来はLSIレイアウトや回路シミュレーションをLinux上のMagicとSpiceを使って実施し, ハードウェア記述言語の部分はSUNのワークステーションで実施していました. 私が散々要求したかいもあってNTTのPARTHENONがLinuxに移植され使えるようになりました. 今年度はまだSUNでも間に合う人数だったのですが, 次からは人数も増えるのでLinux上で実施するつもりです. この連載はRISCアーキテクチャの話もすることになっているので,
これに合わせて
将来的にはRISCプロセッサの設計の話もしたいと思います.
(PARTHENONは残念ながらx86のLinux版しかありませんが, おそらくem86を使ってAlphaでも動作すると思います.)

PARTHENONに興味がある方は

http://www.kecl.ntt.co.jp/car/parthe/index_j.htm

をご参照ください.

Linux版の入手は

http://www.kecl.ntt.co.jp/car/parthe/html/ftpsite.htm

から可能です. 期間限定の暫定版ですが, 十分使いものになります.
私の研究室ではPARTHENONでパイプラインJAVAチップの設計をしています

が，10万ゲートクラスの論理合成にも十分対応できます．

今回はいろいろと忙しい時期にぶつかった上NetBSDとLinuxのデュアルブートを実現しようと施行錯誤が多かったのでスケジュールが厳しくなってしまいました．そこでAS200以外のプロセッサに考慮した記事にはなっていないのですがご容赦ください．