8086/680xx用BCPLコード ジェネレーター
アセンブラコードオプティマイザは役立つか？

Ka Gyou Jun　Fujinami Norihisa　Yamauchi Muneo　Kanada Yasumasa　Murao Hirokazu
何暁旬＊　藤波順久＊　山内宗夫＊＊　金田康正＊＊＊　村尾裕一＊＊＊
＊：東大理学系研究科　＊＊：東大工学系研究科　＊＊＊：東大大型計算機センター

はじめに

今流行のC言語の祖母（祖父？）に当たる言語ＢＣＰＬ〔1〕は，日本ではまったくポピュラーでない。因みにCの母（父？）かつＢＣＰＬの子供に当たる言語Ｂは４．３ＢＳＤについてきている。しかし非常に少数ではあるがその言語のスマートさ，Syntactic sugarの良さ等から熱烈なファンが日本にもいる。世界的にはＢＣＰＬを開発したMartin Richardsがいるケンブリッジ大学並びに，その卒業者が多くいる所，例えばＣＥＲＮ，カナダ・ブリティシュコロンビア大学のある研究室，等ではＢＣＰＬはC以上に使われている。米国コモドール社の６８０００を用いたＡＭＩＧＡのＯＳのもとになったＴＲＩＰＯＳ〔2〕と呼ばれるＯＳ，マクロプロセッサー〔3〕等がＢＣＰＬでかかれていることから，それなりの地位を保っていることが分かる。さらに驚くべき事に，イギリスにおけるＴｈｅ　ＰＣと言えるＢＢＣマイクロコンピュータ（ＣＰＵは６５０２）の上でもＢＣＰＬが動く〔4〕ということである。（値段はマニュアルも含め約６０ポンド，日本円で1万5千円程度。マニュアル代はその約１／４。）

今回このような，日本及び米国では全くマイナーなＢＣＰＬコンパイラ（出力はオブジェクトコードにするべきであったが，人手の関係で今の所出力はアセンブラコード）を日立Ｅ７３００・Ｕｎｉｘシステム（ＣＰＵは６８０２０），ＭＳ－ＤＯＳが動く機械（ＣＰＵは８０８６あるいはそれの上位コンパチブルの物）用に作った。－ＭＳ－ＤＯＳ用のアセンブラフォーマットはほぼＭＡＳＭフォーマット１種と言えるので問題はない。しかし６８０ｘｘ・Ｕｎｉｘにおいては，アセンブラフォーマットは各種存在する為に，Ｅ７３００とは異なるＵｎｉｘシステムへの移植はアセンブラフォーマットの変換が必要であるが，その変更は簡単にすむように工夫しているのでそれほど時間はかからないはずである。－さらに出力であるアセンブラコードのオプティマイザを書き，その最適化効率を調べ，比較のためにCコンパイラのコード効率と比較して見た。第１章でＢＣＰＬの簡単な説明と，今回の試みの動機を，第２章で現在のＢＣＰＬコンパイラの仕組み，第３章でアセンブラコードの最適化前後の比較，第４章で今回の方法でやり残していることを中心に述べる。

1．　ＢＣＰＬとは

ＢＣＰＬはMarchin Richardsが１９６７年に設計した，主にシステムプログラム記述用の言語である。どのような計算機言語にもそれなりの設計哲学があるように，ＢＣＰＬにもそれがある。（詳しくは文献〔1〕を参照。）以下にその特徴を簡単に記す；

◎データタイプはただひとつ。
◎ＢＣＰＬがモデルとする計算機のメモリーはワード（このワードのビット長は計算機によって異なる）の連続であり，そのワードにはどのようなビットパターンもしまうことができる。メモリーにしまわれたデータ，ビットパターン，をアドレスと思うか整数と思うか，あるいは真偽値等，と思うかはプログラマーの判断による。
◎第２章で説明するようにコンパイラの移植性が良い。
◎モジュール間のデータのやり取り，あるいは関数のエントリーアドレスをしまう為のグローバルベクターと言うスタティックな記憶が用意してあり，それを利用することで分割コンパイルができる。
◎Syntactic Sugarが豊富に用意してあること。例えば，コメントの表記方法が６種類もある。また文あるいは式の区切れとなる　；　やＤＯ，ＴＨＥＮと言った予約記号／文字は，明らかに文脈から省略が可能な場合，それは省略できる。Ｅｔｃ．Ｅｔｃ．
このような特徴の為に根強いファンがいるのであろう。今日本で流行のＣと比較して，同じ血筋を引く言語と言っても，特に「豊富な制御構造や，文脈から明らかに省略可能な区切り記号／文字の省略」といった親の良いところをなぜ受けつがなかったのかと思うのは，筆者達ばかりではあるまい。

今回作ったＢＣＰＬコンパイラはアセンブラコードを出力するが，コンパイラがほとんど最適化を行っていないので，第２章で述べるように，入力式のパターンによって決まる，一定の構造をもつアセンブラコードが出力される。そこで考えたことは，

「決まったパターンでアセンブラが出力されるならば，レジスタの使い方，間接アドレスの計算パターンも決まっているはず。－＞ラベルとラベルの間に直接飛び込むことはない事が保証されているので，定型パターンを見出し，別のパターンにおきかえる形の最適化は，簡単に書けるのではないか。そうすれば，わざわざＣＰＵ依存の最適化部分をコンパイラの中に埋めこむ事もなく，新しいＣＰＵの上で動く最適化コンパイラが作れる。そして最適化フェーズがコンパイラ本体とは独立な為に，最適化ルーチンのデバッグは簡単になり，しかもＢＣＰＬの移植性の良さを保つことができるはずである。」

と言う事である。このようにして機種毎に最適化ルーチンを作ってしまうと，最適化ルーチンは全くポータブルで無くなってしまいＢＣＰＬの移植性の良さは失われてしまう。しかしこれは，最適化用のパターンの記述が簡単に行えるような最適化ルーチンを作る事で解決できる。

２．　ＢＣＰＬコンパイラの仕組み

ＢＣＰＬコンパイラは以下の図のように字句解析を行いながら構文解析を施し，ＡＥ木（Applicative Expression Tree）と呼ばれる木構造をメモリー上に作る。その後，この

ＡＥ木をたどりながら翻訳フェーズがＯＣＯＤＥと呼ばれる中間コードの命令を生成する。そしてコードジェネレータがさらにＯＣＯＤＥの翻訳を行い，目的とする計算機のオブジェクトコードあるいはアセンブリコードあるいはブートストラップ用に設計されたもうひとつの中間コードＩＮＴＣＯＤＥに落とすのである。

ＢＣＰＬソース　－＞　字句解析及び構文解析　－＞　ＡＥ木　－＞
翻訳フェーズ　－＞　ＯＣＯＤＥ　－＞　コード生成　－＞目的コード

ＢＣＰＬコンパイラがわざわざ中間コードを経由して目的コードを出すのは，移植性を考慮しての事である。実際すでに２５種類以上の計算機にＢＣＰＬが移植されている事〔1〕を考えると，この方法は成功していると言える。なおＯＣＯＤＥはＢＣＰＬコンパイラ用の抽象的なアセンブリ言語であり，スタックマシンをシミュレートしていると考えてよい。（ＯＣＯＤＥの詳しい説明は文献〔5〕にあるので興味のある人は参照されたい。）またブートストラップ用に設計されたＩＮＴＣＯＤＥは，８個の命令しかない仮想ＩＮＴＣＯＤＥマシン用のアセンブラ言語である。このＩＮＴＣＯＤＥマシンは仕様が簡単な為に，移植しようとする機械の上でＩＮＴＣＯＤＥインタープリタを書く事は容易である。従って，ＢＣＰＬコンパイラシステムをＩＮＴＣＯＤＥの形で用意し，ＩＮＴＣＯＤＥインタープリタを用意すれば，新しい機械のうえでＢＣＰＬコンパイラがすぐに動くことになる。（ＩＮＴＣＯＤＥインタープリタで動くのでネイティブコードで動く場合と比較し約１０倍遅くなる。）そうすればクロスシステムを用いなくても，ＢＣＰＬコンパイラの移植が目的とする計算機の上で行えるようになり，移植作業は楽になる訳である。このＩＮＴＣＯＤＥを用いたＢＣＰＬの移植については文献〔6〕が参考になる。

今回我々が手を入れたＢＣＰＬコンパイラは，電気通信大学・武市研究室から東京大学大型計算機センター木村氏の手を経て入手した物である。武市研究室では，ＭＥＬＣＯＭ－ＣＯＳＭＯ機の上で動くＢＣＰＬコンパイラを作り，木村氏はそれをもとにＶＡＸ－Ｕｎｉｘで動くようにした。我々は，そのソースをもとに，６８０ｘｘ及び８０８６の上で動くようにした訳である。従って，コードジェネレータを別にすれば，基本部分（構文解析及び翻訳フェーズ）はほぼ同じと言って良いので，ＭＥＬＣＯＭ－ＣＯＳＭＯやＶＡＸ－Ｕｎｉｘの上で動くプログラムは，ワード長の違い，割り算における商と余りの符号の付き方の違いを別にして，そのまま６８０ｘｘと８０８６の上でも動く事になる。

以下に，今回作成した６８０００及び８０８６ＢＣＰＬコンパイラのアセンブラ出力例，並びにどのようなＢＣＰＬソースパターンがどのようなアセンブラパターンになるかを６８０００／６８０２０の場合について示す。ＢＣＰＬコンパイラはそれほど複雑な事を行っていない事がお分かりいただけるでしょう。

```
        include SMALL.INC
        EXTRN   _bcplentry:NEAR,_bcplexit:NEAR
_TEXT   SEGMENT
        ASSUME  CS:_TEXT,SS:DGROUP,DS:DGROUP,ES:NOTHING
        EVEN
_bcpl   PROC    NEAR
L0:
        jmp     L2
;
; START
;
        public _BCPL_START
_BCPL_START:
L1:
        pop     bx
        call    _bcplentry
        lea     ax,-16386[si]
        shr     ax,1
        mov     -2[si],ax
        mov WORD PTR -16388[si],0
        mov WORD PTR -16390[si],0
        mov WORD PTR -16392[si],0
        lea     ax,DGROUP:L999
        shr     ax,1
        mov     -16396[si],ax
        sub     si,16400
        call    WORD PTR DGROUP:BASE_GLOBAL[120]
        mov WORD PTR -16394[si],0
        jmp     L3
L4:
        mov WORD PTR -16398[si],46
        sub     si,16402
        call    WORD PTR DGROUP:BASE_GLOBAL[28]
        mov WORD PTR -16388[si],2
        mov WORD PTR -16390[si],0
        mov WORD PTR -16396[si],1
        mov WORD PTR -16398[si],8191
        jmp     L5
L6:
        mov     ax,-1
        mov     -16400[si],ax
        mov     ax,-16396[si]
        add     ax,-2[si]
        shl     ax,1
        xchg    bx,ax
        mov     ax,-16400[si]
        mov     [bx],ax
        mov     ax,-16396[si]
        inc     ax
        mov     -16396[si],ax
L5:
        mov     ax,-16396[si]
        cmp     ax,-16398[si]
        jg      L998
        jmp     L6
        EVEN
L998:
        xor     ax,ax
        mov     -16396[si],ax
        mov     ax,1
        add     ax,-2[si]
        shl     ax,1
        xchg    bx,ax
        mov     ax,-16396[si]
        mov     [bx],ax
        jmp     L8
L7:
        mov     ax,-16388[si]
        imul    WORD PTR -16388[si]
        mov     -16392[si],ax
        jmp     L10
L9:
        xor     ax,ax
        mov     -16396[si],ax
        mov     ax,-16392[si]
        add     ax,-2[si]
        shl     ax,1
        xchg    bx,ax
        mov     ax,-16396[si]
        mov     [bx],ax
        mov     ax,-16392[si]
        add     ax,-16388[si]
        mov     -16392[si],ax
L10:
        mov     ax,-16392[si]
        cmp     ax,8191
        jg      L997
        jmp     L9
        EVEN
L997:
L11:
        mov     ax,-16388[si]
        inc     ax
        mov     -16388[si],ax
        mov     ax,-16388[si]
        add     ax,-2[si]
        shl     ax,1
        xchg    bx,ax
        mov     ax,[bx]
        test    ax,ax
        jnz     L996
        jmp     L11
        EVEN
L996:
L8:
        mov     ax,-16388[si]
        cmp     ax,90
        jg      L995
        jmp     L7
        EVEN
L995:
        mov WORD PTR -16396[si],1
        mov WORD PTR -16398[si],8191
        jmp     L12
L13:
        mov     ax,-16396[si]
        add     ax,-2[si]
        shl     ax,1
        xchg    bx,ax
        mov     ax,[bx]
        test    ax,ax
        jnz     L994
        jmp     L14
        EVEN
L994:
        mov     ax,-16390[si]
        inc     ax
        mov     -16390[si],ax
L14:
        mov     ax,-16396[si]
        inc     ax
        mov     -16396[si],ax
L12:
        mov     ax,-16396[si]
        cmp     ax,-16398[si]
        jg      L993
        jmp     L13
        EVEN
L993:
        mov     ax,-16394[si]
        inc     ax
        mov     -16394[si],ax
L3:
        mov     ax,-16394[si]
        cmp     ax,9
        jg      L992
        jmp     L4
        EVEN
L992:
        mov     ax,-16390[si]
        mov     -16396[si],ax
        lea     ax,DGROUP:L991
        shr     ax,1
        mov     -16398[si],ax
        sub     si,16402
        call    WORD PTR DGROUP:BASE_GLOBAL[152]
        jmp     _bcplexit
        EVEN
L2:
_bcpl   ENDP
_DATA   SEGMENT
        EVEN
L999    LABEL   WORD
        DB      00Fh,031h
        DB      030h,020h
        DB      069h,074h
        DB      065h,072h
        DB      061h,074h
        DB      069h,06Fh
        DB      06Eh,073h
        DB      02Eh,00Ah
L991    LABEL   WORD
        DB      00Ch,00Ah
        DB      025h,04Eh
        DB      020h,070h
        DB      072h,069h
        DB      06Dh,065h
        DB      073h,02Eh
        DB      00Ah,000h
_DATA   ENDS
_GLOBAL         SEGMENT
        ORG     2
        DW      OFFSET  L1
_GLOBAL         ENDS
        EVEN
_TEXT   ENDS
 END
```

８０８６アセンブラ出力例

（入力ソースは付録に示す素数の数を求めるプログラム）

```
  .text
LO:
  jra L2
#
# START
#
  .long 0x05535441
  .long 0x52540000
L1:
  link a6,¥0
  lea -32772(sp),a0
  move.l a0,d0
  lsr.l ¥2,d0
  move.l d0,-4(sp)
  clr.l -32776(sp)
  clr.l -32780(sp)
  clr.l -32784(sp)
  lea L999,a0
  move.l a0,d0
  lsr.l ¥2,d0
  move.l d0,-32792(sp)
  move.l 240(a5),a0
  sub.l ¥32792,sp
  jsr (a0)
  add.l ¥32792,sp
  clr.l -32788(sp)
  jra L3
L4:
  move.l ¥46,-32796(sp)
  move.l 56(a5),a0
  sub.l ¥32796,sp
  jsr (a0)
  add.l ¥32796,sp
  move.l ¥2,-32776(sp)
  clr.l -32780(sp)
  move.l ¥1,-32792(sp)
  move.l ¥8191,-32796(sp)
  jra L5
L6:
  move.l ¥-1,d0
  move.l d0,-32800(sp)
  move.l -32792(sp),d0
  add.l -4(sp),d0
  lsl.l ¥2,d0
  move.l d0,a0
  move.l -32800(sp),(a0)
  move.l -32792(sp),d0
  add.l ¥1,d0
  move.l d0,-32792(sp)
L5:
  move.l -32792(sp),d0
  cmp.l -32796(sp),d0
  jle L6
  clr.l d0
  move.l d0,-32792(sp)
  move.l ¥1,d0
  add.l -4(sp),d0
  lsl.l ¥2,d0
  move.l d0,a0
  move.l -32792(sp),(a0)
  jra L8
L7:
  move.l -32776(sp),d0
  muls.l -32776(sp),d0
  move.l d0,-32784(sp)
  jra L10
L9:
  clr.l d0
  move.l d0,-32792(sp)
  move.l -32784(sp),d0
  add.l -4(sp),d0
  lsl.l ¥2,d0
  move.l d0,a0
  move.l -32792(sp),(a0)
  move.l -32784(sp),d0
  add.l -32776(sp),d0
  move.l d0,-32784(sp)
L10:
  move.l -32784(sp),d0
  cmp.l ¥8191,d0
  jle L9
L11:
  move.l -32776(sp),d0
  add.l ¥1,d0
  move.l d0,-32776(sp)
  move.l -32776(sp),d0
  add.l -4(sp),d0
  lsl.l ¥2,d0
  move.l d0,a0
  move.l (a0),d0
  tst.l d0
  jeq L11
L8:
  move.l -32776(sp),d0
  cmp.l ¥90,d0
  jle L7
  move.l ¥1,-32792(sp)
  move.l ¥8191,-32796(sp)
  jra L12
L13:
  move.l -32792(sp),d0
  add.l -4(sp),d0
  lsl.l ¥2,d0
  move.l d0,a0
  move.l (a0),d0
  tst.l d0
  jeq L14
  move.l -32780(sp),d0
  add.l ¥1,d0
  move.l d0,-32780(sp)
L14:
  move.l -32792(sp),d0
  add.l ¥1,d0
  move.l d0,-32792(sp)
L12:
  move.l -32792(sp),d0
  cmp.l -32796(sp),d0
  jle L13
  move.l -32788(sp),d0
  add.l ¥1,d0
  move.l d0,-32788(sp)
L3:
  move.l -32788(sp),d0
  cmp.l ¥9,d0
  jle L4
  move.l -32780(sp),-32792(sp)
  lea L998,a0
  move.l a0,d0
  lsr.l ¥2,d0
  move.l d0,-32796(sp)
  move.l 304(a5),a0
  sub.l ¥32796,sp
  jsr (a0)
  add.l ¥32796,sp
  unlk a6
  rts
  .align 2
L2:
  .data
  .align 2
L999:
  .long 0x0F313020
  .long 0x69746572
  .long 0x6174696F
  .long 0x6E732E0A
L998:
  .long 0x0C0A254E
  .long 0x20707269
  .long 0x6D65732E
  .long 0x0A000000
  .text
  lea L1,a0
  move.l a0,_gvec+4
  .data
```

68000アセンブラ出力例

（入力ソースは付録に示す素数の数を求めるプログラム）

BCPLのコード生成パターン

D,D6,D7... : データレジスタ.
A,A4 ..... : アドレスレジスタ.
E,E1,E2 .... : 式.
C,C1,C2 .... : ブロック.
n ..... : ループ カウント.

式:

```
E1 + E2   -> move.l E1,D
             add.l  E2,D

E1 - E2   -> move.l E1,D
             sub.l  E2,D

E1 / E2   -> move.l E1,D
             divs.l E2,D

E1 rem E2 -> move.l E1,D
             move.l E2,D6
             divs.l D6,(D7,D)
             move.l D7,D

E1 * E2   -> move.l E1,D
             mult.l E2,D

E1 | E2   -> move.l E1,D
             or.l E2,D

E1 & E2   -> move.l E1,D
             and.l E2,D

E1 equ E2 -> move.l E1,D
             move.l E2,D6
             eor.l D6,D
             not.l D

E1 neqv E2 -> move.l E1,D
              move.l E2,D6
              eor.l D6,D

not E      -> move.l E,D
              not.l D

- E        -> move.l E,D
              neg.l D

E1!E2      -> move.l E1,D
              add.l E2,D
              lsl.l $2,D
              move.l D,A

!E         -> move.l E,D
              lsl.l $2,D
              move.l D,A
```

文:

代入文:

```
E1:=E  -> move.l E,D
          move.l D,E1
```

条件文:

```
1: if E then C1   -> Eは 条件式ならば :              Eは 真理値ならば
2: unless E do C1      ( E1 と E2 から 成る)          ( true : −1 ; false: 0 )
3: test E then C1      move.l E1,D                   move.l E,D
          else C2      cmp.l E2,D                    tst.l D
                       偽 なら 条件 JUMP 命令 L       偽 なら 条件JUMP命令 L
                       (unless の時 真)               (unless の時 真)
                        { C1 }                       { C1 }
                      L:                            (unless の時 真)
                        { C2 }                      L:
                                                     { C2 }
```

繰り返し文:

```
1: while E do C  ->  Eは 条件式ならば:              Eは 真理値ならば
2: until E do C      (E1 と E2 から 成る)           ( true: −1 ; false: 0 )
                     L2:                            L2:
                       { C }                          { C }
                     L:                              L:
                       move.l E1,D                     move.l E,D
                       cmp.l E2,D                      tst.l D
                       真なら 条件JUMP命令 L2          真なら 条件JUMP命令 L2
                       (until の時 偽 なら)            (until の時 偽 なら)

3: C repeatwhile E ->    Eは 条件式ならば:           Eは 真理値ならば
4: C repeatuntil E       (E1 と E2 から 成る)        ( true: −1 ; false: 0 )
                         L:                           L:
                           { C }                        { C }
                         move.l E1,D                  move.l E,D
                         cmp.l E2,D                   tst.l D
                         真なら 条件JUMP命令 L        真なら 条件JUMP命令 L
                         (repeatuntil の時 偽 なら)   (repeatuntil の時 偽 なら)

5:   C repeat       ->      L:
                              { C }
                              jra L

6:  for n=E1 to E2 by K do C ->  move.l E1,D
                                 move.l D,n
                                 move.l n,D
                                 add.l K,D
                                 move.l D,n
                                 move.l n,D
                                 cmp.l E2,D
                                 偽 なら 条件JUMP命令 L
                                   { C }
                                   ........
                                 L:
```

BCPLソースパターンと対応する

68000／68020アセンブラ出力例

ここで，我々は，ただコードジェネレータを６８０ｘｘと８０８６用に新しく作っただけではなく，以下に示す機能強化をＭ－シリーズ上の最適化ＢＣＰＬコンパイラ，ＢＣＰＬ－Ｘ，を参考に行った事を付記しておく。

◎ＡＳＭ命令を用いて，アセンブラ語を出力中に埋めこめるようにした事。

◎完全ではないが一応ＢｒａｎｃｈのＢｒａｎｃｈを取る事。

◎定数式をコンパイル時に評価すること。

◎条件コンパイル機能。

さらに別の機能強化も幾つか試みているが，この原稿を書いている時点でまだデバッグが完了していないのでここには記さない。

３．　アセンブラコードの最適化結果

アセンブラコードの最適化にあたって，次に示すパターンのみを処理するようにしてある。

番号	変換前の命令	ｸﾛｯｸ	変換後の命令	ｸﾛｯｸ	注意
0	mov ax,〔x〕	11/10			ﾚｼﾞｽﾀはないと仮定
1	**p ax,i	4	**p 〔x〕,i	18	ﾚｼﾞｽﾀはないと仮定
2	mov 〔x〕,ax	9			
0	mov ax,〔x〕	11/10	mov ax,〔y〕	11/10	ﾚｼﾞｽﾀはないと仮定
1	**p ax,〔y〕	11	**p 〔x〕,ax	16	
2	mov 〔x〕,ax	9			
0	mov 〔x〕,ax	9	mov 〔x〕,ax	9	
1	mov ax,〔x〕	11/10			
0	mov ax,i	4			ﾚｼﾞｽﾀはないと仮定
1	--p ax,〔x〕	11	--p 〔x〕,i	18	ﾚｼﾞｽﾀはないと仮定
2	mov 〔x〕,ax	9			
0	--p ax,〔x〕	11	--p 〔x〕,ax	16	ﾚｼﾞｽﾀはないと仮定
1	mov 〔x〕,ax	9			
0	mov ax,*x		mov bx,*x		
1	**p ax,*y		**p bx,*y		
2	xchg bx,ax	3			
0	mov ax,*x		mov bx,*x		
1	**p ax,*y		**p bx,*y		
2	**q ax,*z		**q bx,*z		ｼﾌﾄを想定
3	xchg bx,ax	3			
0	mov 〔w〕,ax	9			
1	mov ax,*x		mov bx,*x		
2	**p ax,*y		**p bx,*y		
3	xchg bx,ax	3			
4	mov ax,〔w〕	11/10			
0	mov 〔w〕,ax	9			
1	mov ax,*x		mov bx,*x		
2	**p ax,*y		**p bx,*y		
3	**q ax,*z		**q bx,*z		ｼﾌﾄを想定
4	xchg bx,ax	3			
5	mov ax,〔m〕	11/10			

余滴：

下の言葉は中国語の計算機用語です。

さて、日本語では何にあたるでしょうか。

- 優化
- 汇编语言
- 代码
- 模块

i, j, ...:　定数ax,bx,...:ﾚｼﾞｽﾀ〔x〕,〔y〕,...:ﾒﾓﾘ
*x, *y, ...:　適当なｵﾍﾟﾗﾝﾄﾞ
**p, **q, ...:　第1ｵﾍﾟﾗﾝﾄﾞを変化させる命令
下のもののほか、sub,inc,dec,shl,shr,lea
--p, --q, ...:　交換法則の成り立つ演算命令
add, and, or, xor

８０８６の場合

最適化対象となるアセンブラコードパターン

68000／68020の場合
最適化対象となるアセンブラコードパターン

変換可能なパターン
D ..... : データレジスタ.
A ..... : アドレスレジスタ.
R**.... : 演算命令(add.l, sub.l, and.l, neg.l, not.l, or.lを含む)
*x,*y... : 適当なオペランド.
$i,$j... : 定数値.

変換前命令		変換後の命令
move.l D,*x move.l *x,D	->	move.l D,*x
move.l *x,D move.l D,*y	->	move.l *x,*y
move.l *x,D R** $i,D move.l D,*x	* move.l $i,D or R** *x,D -> move.l D,*x	R** $i,*x
move.l *x,D R** $i,D move.l D,*y	* move.l $i,D or R** *x,D -> move.l D,*y	move.l *x,*y R** $i,*y
move.l *x,D R** *y,D move.l D,*x	* move.l *x,D or R** *y,D -> move.l D,*y *(もし R**=sub.l なら次に neg.l を 加える)	move.l *x,D R** D,*y
move.l $i,D R** $j,D ..... move.l D,*x	($k:=$i R** $j ...) ->	move.l $k,D ..... move.l D,*x
move.l *y,D6 eor.l D6,D move.l D,*y	move.l *y,D6 or eor.l D6,D -> move.l D,*x	eor.l D,*y (or move.l *y,*x eor.l D,*x)
move.l *y,D6 eor.l D6,D not.l D move.l D,*y	move.l *x,D6 or eor.l D6,D -> not.l D move.l D,*y	eor.l D,*y not.l *y (or move.l *x,*y eor.l D,*y not.l *y)
clr.l D add.l *x,D	->	move.l *x,D

変換前命令		変換後の命令
clr.l D move.l D,*x	->	clr.l *x
clr.l D cmp.l *x,D	->	tst.l *x
* move.l $i,D or move.l *x,D cmp.l *x,D cmp.l $i,D *(条件ＪＵＭＰ命令の条件を反転する jgt <-> jle ; jmi <-> jge)	->	cmp.l $i,*x
move.l *x,D tst.l D	->	tst.l *x
jgt (or jle,jmi,jge) L jra L' L:	->	jgt <-> jle L' (jmi <-> jge) L:
(６８０２０ の場合)		
move.l *x,A ..... jsr A or jsr (A)	-> (但し、*x は スタック変数ではない)	jsr *x ..... or jsr (*x)
move.l *x,A jmp A or jmp (A)	->	jmp *x (or jmp (*x))
lsl.l $2,D move.l D,A move.l (A),*x	->	move.l 0(A4,D.l*4),*x
lsl.l $2,D move.l D,A move.l *x,(A)	->	move.l *x,0(A4,D.l*4)

さらにアセンブラ出力中に，ＡＳＭ命令で埋めこまれる本来の定型パターンから外れるアセンブラコード，あるいは最適化を行うと問題が生じるようなパターンについて，最適化処理をバイパスする為の特殊なコメント　;{　と　;}　の組を導入している。（;{　と　;}　で挟まれる部分の最適化処理は行わないようになっている。）

この記述法の導入は，コンパイラのバグと最適化ルーチンのバグのきり分けを容易にするし，構文解析時の解析情報を最適化ルーチンに渡せるというメリットもあり，本コンパイラ並びに最適化ルーチンの開発・デバッグを楽にする。

実際に，この方法の性能を評価するために，付録にある素数の数を求めるプログラム，４種類のソートプログラムについて，各種データを採取したので以下に結果を示す。その中で，メーカー提供のＣコンパイラとの比較もおこなっておいた。これら評価結果から１～２割の性能向上が実現できている事がわかる。こちらが支払う努力に対するメリットが大きく，本稿で述べる方法は非常に有効である事が結論ずけられる。なお測定環境は次のとうりである。

６８０２０：ＨＩＴＡＣＨＩ　Ｅ７３００　Ｕｎｉｘ，主記憶４Ｍｂ，１６Ｍｈ
（最適化ルーチンのプログラム行数はコメントを含め４２７行）

Ｖ３０：　ＮＥＣ　ＶＭ４　ＭＳ－ＤＯＳ，主記憶３８４Ｋｂ，８０８７付，８Ｍｈ
（最適化ルーチンのプログラム行数はコメントを含め４５６行）

Ｅ７３００における結果　　単位：Ｓｅｃ。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort
要素の数	50000個	50000個	50000個	1000個
ＢＣＰＬの最適化前	１５．２	１６．９	８．２	５．１
最適化後	１３．８	１５．０	７．３	４．４
Ｃの最適化前	１２．９	１０．７	６．３	３．７
最適化後	１２．７	１０．５	６．３	３．７

プログラム	繰り返しの回数	ＢＣＰＬの最適化前	ＢＣＰＬの最適化後	Ｃの最適化前	Ｃの最適化後
Ｓｉｅｖｅ	１０回	３．３	２．９	１．５	１．５

Ｅ７３００におけるＣのコンパイラが出すアセンブラの行数

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
最適化前	１６５	１７６	１２２	７８	１１１
最適化後	１４１	１５８	１０５	６８	９６

Ｅ７３００におけるＣのオブジェクト・サイズ　単位：　バイト。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
最適化前	４９６	６２０	３７２	２４４	３９６
最適化後	４６４	６１２	３５６	２３６	３８８

Ｅ７３００におけるＣのコンパイル＋アセンブル＋リンクの時間　単位：Sec.

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
時間	1.7	2.1	1.5	1.1	1.4

Ｅ７３００におけるＢＣＰＬの最適化の時間　単位：　Ｓｅｃ。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
時間	０．３	０．４	０．２	０．２	０．２

Ｅ７３００におけるＢＣＰＬのコンパイル時間　単位：　Ｓｅｃ。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
時間	０．４	０．５	０．４	０．４	０．４

Ｅ７３００におけるＢＣＰＬのコンパイラが出すアセンブラの行数

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
最適化前	２１５	２９７	１９０	１４１	１５４
最適化後	１６６	２２５	１５２	１１２	１２４
コメント行	８	５	８	５	２

Ｅ７３００におけるＢＣＰＬのオブジェクト・サイズ　単位：　バイト。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
最適化前	６９６	１１２８	６２０	４０８	６４８
最適化後	６１６	９６０	５６０	３６０	５６８

Ｅ７３００におけるＢＣＰＬのアセンブル＋リンク時間　単位：　Ｓｅｃ。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
時間	１．１	１．４	１．１	０．８	０．９

PC9801（VM4）における結果　単位：Sec。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort
要素の数	10000個	10000個	10000個	1000個
BCPLの最適化前	11.56	10.12	6.26	28.04
最適化後	10.60	8.13	5.07	21.63
MS C	2.20	6.78	4.33	16.99

プログラム	繰り返しの回数	BCPLの最適化前	BCPLの最適化後	MS C
Sieve	10回	7.60	5.88	4.80

PC9801（VM4）におけるCのコンパイラが出すアセンブラの行数

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
MS C	247	299	205	142	170
コメント行	42	80	37	23	26

PC9801（VM4）におけるBCPLのコンパイラが出すアセンブラの行数

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
最適化前	257	370	235	176	187
最適化後	227	315	208	158	164
コメント行	8	5	8	5	2

PC9801（VM4）におけるCのオブジェクト・サイズ　単位：　バイト。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
MS C	921	907	726	600	656

PC9801（VM4）におけるBCPLのオブジェクト・サイズ　単位：　バイト。

プログラム	Heapsort	Mergesort	Quicksort	Bubblesort	Sieve
最適化前	1128	1325	950	784	754
最適化後	1042	1186	885	728	696

4.　今後の課題

現在行っていないが，６８０ｘｘについて以下のアセンブラコードの最適化によって改善が可能である。（８０８６についてもほぼ同様の事が言える。）さらに，構文解析情報をコメントの形でアセンブラコードに埋め込み，その情報を利用して最適化を行ってみる事，そして最適化が機械に依存しない（パターンの記述が行える）ようにすることは今後の課題となろう。

６８０ｘｘにおけるアセンブラコード最適化項目

◎レジスタの割り当ての最適化

◎ｆｏｒｗａｒｄ　ｓｈｏｒｔ　ｊｕｍｐ　コード

これはあまり有効でないかもしれない。なぜならばループは主としてｂａｃｋｗａｒｄ　ｊｕｍｐとなるから。

◎Ｅ１！Ｅ２：＝？のコード　及び　Ｅ１%Ｅ２：＝？のコードの最適化

現在は，右辺の値を一度スタックに保存してから左辺のアドレス計算を行うようになっている。右辺の値をスタックに保存する必要がない場合が多いので，この部分が除けるはずである。

◎以下のコードの最適化

```
1 :  move.l *x,D
     R1** $i,D          move.l *x,D
     R2** *y,D   ->     R1** $i,D
     move.l D,*y        R2** D,*y
          ( R2**=sub.lならneg.lを加える)
```

```
2 :  move.l *x,D
     R1** $i,D          R1** $i,*x
     R2** *y,D   ->     move.l *y,D
     move.l D,*x        R2** D,*x
          ( R1** and R2**≠sub.lなら)
3 :  move.l $i,D
     R1** *x,D          R1** $i,*x
     R2** *y,D   ->     move.l *x,D
     move.l D,*y        R2** D,*y

4 :  move.l $i,D
     R1** *x,D          R1** $i,*x
     R2** *y,D   ->     move.l *y,D
     move.l D,*x        R2** D,*x
```

5．　おわりに

ここで述べた最適化手法の主眼は，

「翻訳フェーズ及びコードジェネレータはバカに徹し，アセンブラコードを出すべきである。そうすれば出てきたアセンブラコードは定型パターンとなるので，パターンマッチで最適化を行う手法が有効になる。この方法で言語処理系を作成する方法は，コンパイラシステムを完成させるまでの開発期間を短縮し，又出来上がった言語処理系は十分実用に耐えるものとなる。」

と言うことである。この方法を適応するとコンパイラの作成は手軽になり，生成される目的コードも決して悪くないものができる。今後この方法を新しい処理系の設計，作成に用いる予定である。

又，ＢＣＰＬで書かれたＲＯＦＦ，エディター，マクロ処理系等テキスト処理系が手元にあるので６８０ｘｘ及び８０８６へ移植を行い，利用するとともに，大型計算機センターを通じて公開する予定でいる。その為には，最適化の徹底と移植性の向上，並びに実行時ルーチンの改善及びルーチンサイズの縮小を早急に行う必要がある。

最後に，今回２種類のＣＰＵにＢＣＰＬコンパイラを移植したが，その作業時間からすると，ＢＣＰＬを新しいＣＰＵに移植しようとした時；

◎ＢＣＰＬと新しいＣＰＵの両方の知識があると・・・約１週間

◎どちらか片方の知識しかない人同士組むと・・・・・約２週間

◎片方の知識もない時・・・・・・・・・・・・・・・約１か月

の期間が必要のように感じられた。この期間はＶ６０／７０あるいはトランスピュータ等のＣＰＵへＢＣＰＬを移植するときにはどのようになるのか，大変興味のあるところである。

Why don't you like BCPL?

文献

〔1〕：M. Richards & C. Whitby-Strevens, "BCPL, The language and its compiler," Cambridge University Press, 1979,和田英一訳，「BCPL 言語とそのｺﾝﾊﾟｲﾗ」,共立出版社.

〔2〕：M. Richards, A. R. Aylward, P. B. Bond, R. D. Evans & B. J. Knight, "TRIP OS - A portable operating system formini-computers," Software - Proactice and Experience, 9, 513-526(1979).

〔3〕：P. J. Brown, "SUPERMAC - A Macro Facility that can be Added to Existing Compilers," ibid, 10, 431-434(1980).

〔4〕：C. Jobson & J. Richards, "BCPL for the BBC Microcomputer," User Guide, Acornsoft Ltd., 1983.

〔5〕：M. Richards, "The portability of the BCPL compiler," Software - Practice and Experience, Vol. 1, No. 2 (1971)

〔6〕: M. Richards, "Bootstrapping the BCPL compiler using Intcode," Van der Poel W. L. & Maarssen L. A. (eds.)Machine-Oriented Higer Level Languages, North-Holland, 1974, pp. 265-270

付録　ベンチマークに使用したプログラムリスト

```
SECTION "SIEVE"

GET "LIBHDR.HDR"

MANIFEST $(
       MAX=8191
       RMAX=90
$)

LET START() BE $(
       LET A=VEC MAX
       LET P,N,I=0,0,0
       WRITES("10 iterations.*N")
       FOR C=0 TO 9 $(
              WRCH('.')
              P:=2 N:=0
              FOR I=1 TO MAX A!I:=TRUE
              A!1:=FALSE
              WHILE P<=RMAX $(
                     I:=P*P
                     WHILE I<=MAX $( A!I:=FALSE I:=I+P $)
                     P:=P+1 REPEATUNTIL A!P
              $)
              FOR I=1 TO MAX IF A!I N:=N+1
       $)
       WRITEF("*NX%N primes.*N",N)
$)
```

```
SECTION "SORT1"

GET "LIBHDR.HDR"

//    Heap sort

MANIFEST
$( MEMBER=9999
   SEED=7415
   P=25543
                              $)
GLOBAL
$( J : NEXTGLOBAL    ; K: NEXTGLOBAL+1
   A : NEXTGLOBAL+2
                                   $)
 LET START() BE
 $( LET V=VEC MEMBER
    A:=V
    A!0:=(12345+P) REM SEED
    FOR I=0 TO MEMBER-1 DO $( I:=I+1;A!I:=(A!(I-1)+P) REM SEED $)
// Incorrect translation from C I:=I+1; must be deleted!
// However, correct code was used for the E7300 test.
    J:=MEMBER/2+1
    K:=MEMBER
    WHILE J>0 DO $( J:=J-1 ; SIFT()      $)
    WHILE K>0 DO $( LET S=A!0
                    A!0:=A!K ; A!K:=S
                    K:=K-1 ; SIFT()      $)
    WRITEF("Heap sort finished %n *N",MEMBER+1)      $)

AND SIFT() BE
$( LET L=J
   AND S,I=A!J,J*2
   WHILE I<=K DO
   $( IF (I<K) & (A!I<A!(I+1)) THEN I:=I+1
      IF S>=A!I THEN GOTO LAB
      A!L,L,I:=A!I,I,2*L      $)
 LAB: A!L:=S      $)
```

```
SECTION "SORT3"

GET "LIBHDR.HDR"

//    Quick sort

MANIFEST
$( MEMBER=9999
   SEED=7415
   P=25543
                              $)
GLOBAL
$( A : NEXTGLOBAL
   S : NEXTGLOBAL+1
                                   $)
 LET START() BE
 $( LET V=VEC MEMBER
    A:=V
    A!0:=(12345+P) REM SEED
    FOR I=0 TO MEMBER-1 DO $( I:=I+1 ; A!I:=(A!(I-1)+P) REM SEED $)
// Incorrect translation from C I:=I+1; must be deleted!
// However, correct code was used for the E7300 test.
    SORT(0,MEMBER)
    WRITEF("Quick sort finished %N*N",MEMBER+1)      $)


AND SORT(I,J) BE
$( LET L,R=I,J
   S:=A!((J+I)/2+1)
   $( WHILE A!L<S DO L:=L+1
      WHILE A!R>S DO R:=R-1
      IF L<=R THEN
      $( LET P=A!L
         A!L:=A!R ; L:=L+1
         A!R:=P   ; R:=R-1        $)   $) REPEATWHILE L<=R
   IF I<R THEN SORT(I,R)
   IF L<J THEN SORT(L,J)       $)
```

```
SECTION "SORT2"

GET "LIBHDR.HDR"

//    Merge sort

MANIFEST
$( MEMBER=9999
   SEED=7415
   P=25543
                              $)
GLOBAL
$( A : NEXTGLOBAL
                                   $)
 LET START() BE
 $( LET V=VEC MEMBER*2+1
    AND UP,P=TRUE,1
    AND J,K,I,L,R,Q,T=0,0,0,0,0,0,0
    A:=V
    A!0:=(12345+P) REM SEED
    FOR I=0 TO MEMBER-1 DO $( I:=I+1 ; A!I:=(A!(I-1)+P) REM SEED $)
// Incorrect translation from C I:=I+1; must be deleted!
// However, correct code was used for the E7300 test.
    UNTIL P>MEMBER DO
    $( LET H,M=1,MEMBER+1
       TEST UP THEN
       $( I:=0 ; J:=MEMBER ; K:=MEMBER+1 ; L:=2*MEMBER+1   $)
          ELSE $( K:=0 ; L:=MEMBER ; I:=MEMBER+1 ; J:=2*MEMBER+1 $)
       UNTIL M=0 DO
       $( TEST M>=P THEN Q:=P ELSE Q:=M ; M:=M-Q
          TEST M>=P THEN R:=P ELSE R:=M ; M:=M-R
          WHILE (Q$=0) & (R$=0) DO
            TEST A!I<A!J THEN
            $( A!K:=A!I ; K:=K+H
               I:=I+1   ; Q:=Q-1      $)
             ELSE $( A!K:=A!J ; K:=K+H
                     J:=J-1 ; R:=R-1     $)
          WHILE R$=0 DO
          $( A!K:=A!J ; K:=K+H
             J:=J-1   ;  R:=R-1       $)
          WHILE Q$=0 DO
          $( A!K:=A!I ; K:=K+H
             I:=I+1 ; Q:=Q-1         $)
          H:=-H ;  T:=K ; K:=L ; L:=T          $)
       UP:=NOT UP   ; P:=2*P         $)
    IF NOT UP THEN FOR I=0 TO MEMBER DO A!I:=A!(MEMBER+1+I)
    WRITEF("Merge sort finished  %N *N",MEMBER+1)       $)
```

```
SECTION "SORT4"

GET "LIBHDR.HDR"

//    Bubble sort

MANIFEST
$( MEMBER=9999
   SEED=7415
   P=25543                    $)

GLOBAL
$( A : NEXTGLOBAL                   $)

 LET START() BE
 $( LET V=VEC MEMBER
    A:=V
    A!0:=(12345+P) REM SEED
    FOR I=0 TO MEMBER-1 DO $( I:=I+1 ; A!I:=(A!(I-1)+P) REM SEED $)
// Incorrect translation from C I:=I+1; must be deleted!
// However, correct code was used for the E7300 test.
    FOR I=1 TO MEMBER DO
     FOR J=MEMBER TO I BY -1 DO
        IF A!(J-1)>A!J THEN
        $( LET X=A!J
           A!J:=A!(J-1)  ; A!(J-1):=X     $)
    WRITEF("Bubble sort finished %N*N",MEMBER)      $)
```

```
#include <stdio.h>

#define MAX 8191
#define RMAX 100

main()
{
        int a[MAX+1];
        register int p,i;
        int n,c;
        puts("10 iterations.");  fflush(stdout);
        for(c=0;c<10;c++) {
                putchar('.');
                p=2; n=0;
                for(i=1;i<=MAX;i++) a[i]=1;
                a[1]=0;
                while(p<=RMAX) {
                        for(i=p*p;i<=MAX;i+=p) a[i]=0;
                        do p++; while(!a[p]);
                }
                for(i=1;i<=MAX;i++) if(a[i]) n++;
        }
        printf("$n %d primes.$n",n);  fflush(stdout);
}
```

```
/* Bubble sort   */
#define MEMBER 1000
#define SEED 7415
#define P 25543

int a[MEMBER];
main ()
{
        int i,j,x;
        a[0] = (12345+P) % SEED;
        for(i = 1; i < MEMBER; i++) a[i] = (a[i-1] + P) % SEED;
        for(i = 1; i < MEMBER; i++) {
                for(j = MEMBER -1; j >= i; j--) {
                        if(a[j-1] > a[j]) {
                                x = a[j];
                                a[j] = a[j-1];
                                a[j-1] = x;
                        }
                }
        }
        printf("Bubble sort finished  %d $n",MEMBER);
}
```

```
/* Heap sort   */
#define member 10000
#define seed 7415
#define p 25543
int j,k;
int a[member];

main ()
{
    int i,s;
    a[0]=(12345+p) % seed;
    for (i=1 ;i<member ; i++)  a[i]=rudom(i-1);
    j=member/2+1;
    k=member;
    while (j>0) sift(--j,k);
    while (k>0) {
                    s=a[0] ; a[0]=a[k];a[k]=s;
                    sift(j,--k) ;
                }
    printf("Heap sort finished %d order$n",member);
                }

sift(j,k)
{
    int l,s,i;
    l=j ; s=a[j] ; i=j*2;
    while (i<=k)
    {
        if ((i<k) & (a[i] < a[i+1])) i++;
        if (s>=a[i]) goto lab;
        a[l]=a[i] ;
        l=i;
        i=2*l;
                }
 lab: a[l]=s;
                        }
rudom(i)
{
  int z;
  z=(a[i]+p) % seed;
  return(z) ;
     }
```

```
/* Merge sort */
#include <stdio.h>

#define MEMBER 10000
#define SEED 7415
#define P 25543

int a[2 * MEMBER];

main()
{
        int e,up,i;
        a[0] = (12345 + P) % SEED;
        up = 0 ;
        e = 1 ;
        for(i = 1; i < MEMBER; i++)
                a[i] = (a[i-1] + P) % SEED;
        while (e <= MEMBER)   {
                int h,m,j,k,r,q,t,l;
                h = 1;
                m = MEMBER;
                if (up == 0) {
                        i = 0 ;
                        j = MEMBER - 1;
                        k = MEMBER;
                        l = 2 * MEMBER;
                }
                else  {
                        k = 0 ;
                        l = MEMBER - 1;
                        i = MEMBER;
                        j = 2 * MEMBER;
                }
                while (m > 0) {
                        if (m >= e)
                                q = e;
                        else
                                q = m;
                        m -= q;
                        if (m >= e)
                                r = e ;
                        else
                                r = m;
                        m -= r;
                        while ((q != 0)&&(r != 0)) {
                                if (a[i] < a[j]) {
                                        a[k] = a[i];
                                        k += h;
                                        i++;
                                        q--;
                                }
                                else {
                                        a[k] = a[j];
                                        k += h;
                                        j--;
                                        r--;
                                }
                        }
                        while (r != 0) {
                                a[k] = a[j];
                                k += h;
                                j--;
                                r--;
                        }
                        while (q != 0) {
                                a[k] = a[i];
                                k += h;
                                i++;
                                q--;
                        }
                        h *= -1;
                        t = k;
                        k = l;
                        l = t;
                }
                up ^= 1;
                e *= 2;
        }
        if (up != 0)
                for (i = 0; i < MEMBER; i++)
                        a[i] = a[MEMBER+i];
        for (i = 0; i < MEMBER; i++)
                printf(" a[%d] %d$n",i,a[i]);
        printf("Merge sort finished %d $n",MEMBER);
}
```

```
/* Quick sort    */
#include <stdio.h>

#define SEED 7415
#define P 25543
#define MEMBER 10000

int a[MEMBER];

main () {
        int i;
        a[0] = (12345 + P) % SEED;
        for(i = 1; i < MEMBER; i++)
                a[i] = (a[i-1] + P) % SEED;
        SORT(0,MEMBER-1);
        for(i = 0; i < MEMBER; i++)
                printf(" A[%d]= %d $n",i,a[i]);
        printf(" Quick sort finished $n");
}
/* procedure SORT         */
SORT(i,j) {

        int l,r,s,p;
        r = j;
        l = i;
        s = a[(j+i) / 2 + 1];
        while(l <= r) {
                while(a[l] < s) l +=1;
                while(a[r] > s) r -=1;
                if(l <= r) {
                        p = a[l];
                        a[l] = a[r];
                        l++;
                        a[r] = p;
                        r--;
                }
        }
        if(i < r) SORT(i,r);
        if(j > l) SORT(l,j);
}
```