This file lists the size and speed of all of Axe's commands. It also has notes to clarify details about commands, and has examples to show exactly what source code would correspond to the compiled commands detailed in this file. Most of the command names are taken from the Commands.inc file inlcuded in Axe releases, and some may be slightly cryptic. If you are unsure what the name means, just look at the example code given, and that should demonstrate what Axe source code corresponds to that command. And if you can't find something listed here, then it's not actually a command and takes up no size.

Empty parentheses, brackets, and arguments in the examples mean that, although you'll usually want to put something there, whatever you put there is not actually a part of the command. As such, the command sizes and speeds listed reflect the use of these commands without any arguments, because you can enter any arguments you want. The size and speed of arguments can be determined by the other entries listed in this file.

Under the names of some commands are lines that specify information about calling the command. This means that the command is added as a subroutine and will only exist in your code once, but due to that, there will be some size and speed overhead necessary for calling the command. Some commands can also be called in more than one way.

Accurate as of Axe 0.4.8.

COMMAND NAME	SIZE	SPEED	EXAMPLE	EXAMPLE 2	NOTES ;

;Headers
;--

p_AsmHeader:	2 bytes				

An assembly program compiled with no shell and no code (except for the mandated return) will be n+12 bytes; n=length of name

p_IonHeader:	10 bytes				
p_MOSHeader:	39 bytes				
p_DCSHeader:	51 bytes				
p_APPHeader1:	18 bytes				

;Saving and loading numbers
;--

p_LoadConst:	3 bytes	10 cycles	1	ᴇ1337	
p_LoadVar:	3 bytes	16 cycles	A		
p_Load1ByteConstPtr:	5 bytes	23 cycles	$\{L_1\}$		
p_Load2ByteConstPtr:	3 bytes	16 cycles	$\{L_1\}^r$		
p_Load1ByteExprPtr:	3 bytes	14 cycles	{}		
p_Load2ByteExprPtr:	4 bytes	24 cycles	$\{\}^r$		
p_Load2ByteExprPtrBackward:	4 bytes	24 cycles	$\{\}^{rr}$		
p_SaveVar:	3 bytes	16 cycles	→A		
p_Save1ByteConstPtr:	4 bytes	17 cycles	$\rightarrow\{L_1\}$		
p_Save2ByteConstPtr:	3 bytes	16 cycles	$\rightarrow\{L_1\}^r$		
p_Save1ByteExprPtr:	3 byte	28 cycles	→{}		
p_Save2ByteExprPtr:	5 bytes	41 cycles	$\rightarrow\{\}^r$		
p_Save2ByteExprPtrBackward:	5 bytes	41 cycles	$\rightarrow\{\}^{rr}$		

```
;Optimized Math
;-----------------------------------------------
p_Add1:                    1 byte       6 cycles                +1
       Tied for the smallest way to change a value in Axe

p_Add2:                    2 bytes            12 cycles         +2

p_Add3:                    3 bytes            18 cycles         +3

p_Add254:                  3 bytes            16 cycles         +254

p_Add255:                  2 bytes            10 cycles         +255

p_Add256:                  1 byte             4 cycles          +256
       The absolute smallest and fastest way to change a value in Axe

p_Add257:                  2 bytes            10 cycles         +257

p_Add258:                  3 bytes            16 cycles         +258

p_Add510:                  4 bytes            20 cycles         +510

p_Add511:                  3 bytes            14 cycles         +511

p_Add512:                  2 bytes            8 cycles          +512

p_Add513:                  3 bytes            14 cycles         +513

p_Add514:                  4 bytes            20 cycles         +514

p_Add767:                  4 bytes            18 cycles         +767

p_Add768:                  3 bytes            12 cycles         +768

p_Add769:                  4 bytes            18 cycles         +769

p_Add1024:                 4 bytes            16 cycles         +1024

p_Sub1:                    1 byte       6 cycles                -1
       Tied for the smallest way to change a value in Axe

p_Sub2:                    2 bytes            12 cycles         -2

p_Sub3:                    3 bytes            18 cycles         -3

p_Sub254:                  3 bytes            16 cycles         -254

p_Sub255:                  2 bytes            10 cycles         -255

p_Sub256:                  1 byte       4 cycles                -256
       Also the absolute smallest and fastest way to change a value in Axe

p_Sub257:                  2 bytes            10 cycles         -257

p_Sub258:                  3 bytes            16 cycles         -258

p_Sub510:                  4 bytes            20 cycles         -510

p_Sub511:                  3 bytes            14 cycles         -511

p_Sub512:                  2 bytes            8 cycles          -512

p_Sub513:                  3 bytes            14 cycles         -513

p_Sub514:                  4 bytes            20 cycles         -514

p_Sub767:                  4 bytes            18 cycles         -767

p_Sub768:                  3 bytes            12 cycles         -768

p_Sub769:                  4 bytes            18 cycles         -769
```

p_Sub1024:	4 bytes	16 cycles	-1024
p_MulN1:	6 bytes	24 cycles	*-1
The same as p_IntNeg			
p_Mul2:	1 byte	11 cycles	*2
Tied for the smallest way to change a value in Axe			
p_Mul3:	4 bytes	30 cycles	*3
p_Mul4:	2 bytes	22 cycles	*4
p_Mul5:	5 bytes	41 cycles	*5
p_Mul6:	5 bytes	41 cycles	*6
p_Mul7:	6 bytes	52 cycles	*7
p_Mul8:	3 bytes	33 cycles	*8
p_Mul9:	6 bytes	52 cycles	*9
p_Mul10:	6 bytes	52 cycles	*10
p_Mul12:	6 bytes	52 cycles	*12
p_Mul16:	4 bytes	44 cycles	*16
p_Mul32:	5 bytes	55 cycles	*32
p_Mul64:	5 bytes	144 cycles	*64
p_Mul128:	5 bytes	170 cycles	*128
p_Mul255:	6 bytes	31 cycles	*255
p_Mul256:	3 bytes	11 cycles	*256
p_Mul257:	3 bytes	12 cycles	*257
p_Mul258:	4 bytes	23 cycles	*258
p_Mul260:	5 bytes	34 cycles	*260
p_Mul264:	6 bytes	45 cycles	*264
p_Mul512:	4 bytes	22 cycles	*512
p_Mul513:	6 bytes	37 cycles	*513
p_Mul514:	4 bytes	23 cycles	*514
p_Mul516:	5 bytes	34 cycles	*516
p_Mul520:	6 bytes	45 cycles	*520
p_Mul768:	6 bytes	23 cycles	*768
p_Mul1024:	5 bytes	33 cycles	*1024
p_Mul1028:	5 bytes	34 cycles	*1028
p_Mul1032:	6 bytes	45 cycles	*1032
p_Mul2048:	6 bytes	44 cycles	*2048
p_Mul2056:	6 bytes	45 cycles	*2056
p_Mul4096:	5 bytes	290 cycles	*4096
p_Mul8192:	5 bytes	314 cycles	*8192

p_Mul16384:	5 bytes	338 cycles	*16384
p_Mul32768:	6 bytes	24 cycles	*32768
p_Div2:	4 bytes	16 cycles	/2
p_Div10:	3 bytes	~1896 cycles	/10
n*3+1878 cycles, n=number of set bits in result			
p_Div128:	5 bytes	27 cycles	/128
p_Div256:	3 bytes	11 cycles	/256
p_Div512:	5 bytes	19 cycles	/512
p_Div32768:	5 bytes	27 cycles	/32768
p_SDiv2:	4 bytes	16 cycles	//2
p_SDiv64:	6 bytes	38 cycles	//64
p_SDiv128:	4 bytes	23 cycles	//128
p_SDiv256:	5 bytes	20 cycles	//256
p_SDiv512:	6 bytes	38 cycles	//512
p_SDiv16384:	6 bytes	38 cycles	//16384
p_SDiv32768:	3 bytes	26 cycles	//32768
p_Mod2:	5 bytes	20 cycles	^2
p_Mod4:	6 bytes	22 cycles	^4
p_Mod8:	6 bytes	22 cycles	^8
p_Mod16:	6 bytes	22 cycles	^16
p_Mod32:	6 bytes	22 cycles	^32
p_Mod64:	6 bytes	22 cycles	^64
p_Mod128:	4 bytes	15 cycles	^128
p_Mod256:	2 bytes	7 cycles	^256
p_Mod512:	4 bytes	15 cycles	^512
p_Mod1024:	4 bytes	15 cycles	^1024
p_Mod2048:	4 bytes	15 cycles	^2048
p_Mod4096:	4 bytes	15 cycles	^4096
p_Mod8192:	4 bytes	15 cycles	^8192
p_Mod16384:	4 bytes	15 cycles	^16384
p_Mod32768:	2 bytes	8 cycles	^32768
p_EQN512:	9 bytes	44 cycles	=¯512
p_EQN256:	8 bytes	40 cycles	=¯256
p_EQN2:	8 bytes	40 cycles	=¯2
p_EQN1:	7 bytes	36 cycles	=¯1
p_EQ0:	7 bytes	36 cycles	=0

```
p_EQ1:                    7 bytes         ~29 cycles        =1
      24 cycles if true, 34 cycles if false

p_EQ2:                    8 bytes         ~33 cycles        =2
      28 cycles if true, 38 cycles if false

p_EQ256:                  8 bytes         40 cycles         =256

p_EQ512:                  9 bytes         44 cycles         =512

p_NEN512:                 9 bytes         ~31 cycles        ≠¯512
      33 cycles if true, 28 cycles if false

p_NEN256:                 8 bytes         ~27 cycles        ≠¯256
      29 cycles if true, 24 cycles if false

p_NEN2:                   8 bytes         40 cycles         ≠¯2

p_NEN1:                   7 bytes         36 cycles         ≠¯1

p_NE0:                    7 bytes         ~23 cycles        ≠0
      25 cycles if true, 20 cycles if false

p_NE1:                    8 bytes         ~27 cycles        ≠1
      29 cycles if true, 24 cycles if false

p_NE2:                    9 bytes         ~31 cycles        ≠2
      33 cycles if true, 28 cycles if false

p_NE256:                  8 bytes         ~27 cycles        ≠1
      29 cycles if true, 24 cycles if false

p_NE512:                  9 bytes         ~31 cycles        ≠2
      33 cycles if true, 28 cycles if false

p_LE0:                    7 bytes         36 cycles         ≤0

p_LT1:                    7 bytes         36 cycles         <1

p_GE1:                    7 bytes         ~23 cycles        ≥1
      25 cycles if true, 20 cycles if false

p_GT0:                    7 bytes         ~23 cycles        >0
      25 cycles if true, 20 cycles if false

p_SGE0:                   4 bytes         32 cycles         ≥≥0

p_SLT0:                   5 bytes         27 cycles         <<0

p_GetBit0:                5 bytes         27 cycles         ee0

p_GetBit1:                6 bytes         38 cycles         ee1

p_GetBit2:                7 bytes         49 cycles         ee2

p_GetBit3:                8 bytes         ~30 cycles        ee3
      29 cycles if 1, 30 cycles if 0

p_GetBit4:                8 bytes         ~30 cycles        ee4
      29 cycles if 1, 30 cycles if 0

p_GetBit5:                8 bytes         ~30 cycles        ee5
      29 cycles if 1, 30 cycles if 0

p_GetBit6:                7 bytes         26 cycles         ee6

p_GetBit7:                6 bytes         22 cycles         ee7

p_GetBit8:                5 bytes         27 cycles         ee8          e0
```

```
p_GetBit9:                6 bytes             38 cycles           ee9           e1

p_GetBit10:               7 bytes             49 cycles           ee10          e2

p_GetBit11:               8 bytes             ~30 cycles          ee11          e3
       29 cycles if 1, 30 cycles if 0

p_GetBit12:               8 bytes             ~30 cycles          ee12          e4
       29 cycles if 1, 30 cycles if 0

p_GetBit13:               8 bytes             ~30 cycles          ee13          e5
       29 cycles if 1, 30 cycles if 0

p_GetBit14:               7 bytes             26 cycles           ee14          e6

p_GetBit15:               5 bytes             20 cycles           ee15          e7


;Comparing numbers
;-----------------------------------------------
p_IntEqShort:             9 bytes             43 cycles           =255
       0≤Short<255
p_IntEqNShort:            9 bytes             43 cycles           =⁻256
       **CURRENTY UNAVAILABLE** ⁻256≤Short<0
p_IntEqConst:             11 bytes      ~49 cycles                =1337
       48 cycles if true, 49 cycles if false; if 0≤Const<256, p_IntEqShort is used; if
⁻256≤Const<0, p_IntEqNShort is used
p_IntEqVar:               12 bytes            ~59 cycles          =A
       58 cycles if true, 59 cycles if false
p_IntEqExpr:              10 bytes            ~60 cycles          =()
       59 cycles if true, 60 cycles if false

p_IntNeShort:             9 bytes             43 cycles           ≠255
       0≤Short<255
p_IntNeNShort:            9 bytes             43 cycles           ≠⁻256
       **CURRENTLY UNAVAILABLE** ⁻256≤Short<0
p_IntNeConst:             11 bytes            ~44 cycles          ≠1337
       46 cycles if true, 41 cycles if false; if 0≤Const<256, p_IntNeShort is used; if
⁻256≤Const<0, p_IntNeNShort is used
p_IntNeVar:               12 bytes            ~54 cycles          ≠A
       56 cycles if true, 51 cycles if false
p_IntNeExpr:              10 bytes            ~55 cycles          ≠()
       57 cycles if true, 52 cycles if false

p_IntGtConst:             10 bytes            45 cycles           >1337
p_IntGtVar:               11 bytes            55 cycles           >A
p_IntGtExpr:              8 bytes             52 cycles           >()

p_IntGeConst:             9 bytes             50 cycles           ≥1337
p_IntGeVar:               10 bytes            60 cycles           ≥A
p_IntGeExpr:              9 bytes             65 cycles           ≥()

p_IntLtConst:             9 bytes             41 cycles           <1337
p_IntLtVar:               10 bytes            51 cycles           <A
p_IntLtExpr:              9 bytes             56 cycles           <()

p_IntLeConst:             10 bytes            54 cycles           ≤1337
p_IntLeVar:               11 bytes            64 cycles           ≤A
p_IntLeExpr:              8 bytes             61 cycles           ≤()

p_Min:                    9 bytes             ~63 cycles          min(,)
       63 cycles if first argument is smaller, 62 cycles if first argument is larger

p_Max:                    9 bytes             ~63 cycles          max(,)
       63 cycles if second argument is smaller, 62 cycles if second argument is larger
```

```
;Signed Stuff
;-----------------------------------------------
p_SIntGtConst:            15 bytes     77 cycles                 >>1337
p_SIntGtVar:              16 bytes     87 cycles                 >>A
p_SIntGtExpr:             14 bytes     75 cycles                 >>()

p_SIntGeConst:            14 bytes     70 cycles                 ≥≥1337
p_SIntGeVar:              15 bytes     80 cycles                 ≥≥A
p_SIntGeExpr:             14 bytes     85 cycles                 ≥≥()

p_SIntLtConst:            15 bytes     64 cycles                 <<1337
p_SIntLtVar:              16 bytes     74 cycles                 <<A
p_SIntLtExpr:             14 bytes     88 cycles                 <<()

p_SIntLeConst:            15 bytes     74 cycles                 ≤≤1337
p_SIntLeVar:              16 bytes     84 cycles                 ≤≤A
p_SIntLeExpr:             13 bytes     81 cycles                 ≤≤()


;Addition and Subtraction
;-----------------------------------------------
p_AddConst:               4 bytes            21 cycles           +1337
p_AddVar:                 5 bytes            31 cycles           +A
      Smaller and faster than subtraction
p_AddExpr:                3 bytes            42 cycles           +()
      Smaller and faster than subtraction

p_SubConst:               4 bytes            21 cycles           -1337
p_SubConst:               7 bytes            39 cycles           -A
      Larger and slower than addition
p_SubExpr:                6 bytes            54 cycles           -()
      Larger and slower than addition

p_IntNeg:                 6 bytes            24 cycles           ⁻A           ⁻()
      The same as *⁻1

p_AbsInt:                 10 bytes     ~29 cycles                abs()
      20 cycles if positive, 39 cycles if negative

p_MiniSigned:             5 bytes            23 cycles           sign{}


;Bit Operations
;-----------------------------------------------
p_IntOrConst:             9 bytes            34 bytes            + ᴇ1337
p_IntOrVar:               10 bytes           44 bytes            + A
p_IntOrExpr:              8 bytes            45 bytes            + ()

p_IntAndConst:            9 bytes            34 bytes            ·ᴇ1337
p_IntAndVar:              10 bytes           44 bytes            ·A
p_IntAndExpr:             8 bytes            45 bytes            ·()

p_IntXorConst:            9 bytes            34 bytes            □ᴇ1337
p_IntXorVar:              10 bytes           44 bytes            □A
p_IntXorExpr:             8 bytes            45 bytes            □()

p_IntNot:                 6 bytes            24 cycles           not()ʳ

p_BoolOrConst:            6 bytes            22 bytes            or ᴇ37
p_BoolOrVar:              7 bytes            32 bytes            or A
p_BoolOrExpr:             5 bytes            33 bytes            or ()

p_BoolAndConst:           6 bytes            22 bytes            and ᴇ37
p_BoolAndVar:             7 bytes            32 bytes            and A
p_BoolAndExpr:            5 bytes            33 bytes            and ()

p_BoolXorConst:           6 bytes            22 bytes            xor ᴇ37
p_BoolxorVar:             7 bytes            32 bytes            xor A
```

```
p_BoolXorExpr:            5 bytes           33 bytes          xor ()

p_BoolNot:                3 bytes           12 cycles         not()


;Control Structures
;-----------------------------------------------
p_IfTrue:                 5 bytes           18 cycles         If
      Must be paired with an End later in the code

p_IfFalse:                5 bytes           18 cycles         !If
      Must be paired with an End later in the code

p_Else:                   3 bytes           10 cycles         Else
      Must be preceded by an If or !If and followed by an End

p_ElseIfTrue:             5 bytes           18 cycles         ElseIf
            Must be preceded by an If or !If and followed by an End

p_ElseIfFalse:            5 bytes           18 cycles         Else!If
      Must be preceded by an If or !If and followed by an End

p_DS:                     15 bytes    56/72 cycles       DS<(,)
      56 cycles if not zero, 72 cycles if zero; remember to add the duration of the
reset condition for each zero iteration; must be paired with an End later in the code

p_While:                  8 bytes           n*28+18 cycles          While
            n=number of completed iterations; remember to add the duration of the
check condition for each iteration; must be paired with an End later in the code

p_Repeat:                 8 bytes           n*28+18 cycles          Repeat
                  n=number of completed iterations; remember to add the duration of
the check condition for each iteration; must be paired with an End later in the code

p_For:                    20 bytes    n*98+76 cycles          For(A,,)
      n=number of completed iterations; remember to add the duration of the check
condition for each iteration; must be paired with an End later in the code

p_Goto:                   3 bytes           10 cycles         Goto LBL

p_Call:                   3 bytes           17 cycles         sub(LBL)
      Args:               3 bytes           16 cycles

p_CallSaveArgs:           3 bytes           17 cycles         sub(LBLʳ)
      Args:               12 bytes          73 cycles

p_Ret:                    1 byte            10 cycles         Return

p_RetIfTrue:              3 bytes           18 cycles         ReturnIf

p_RetIfFalse:             3 bytes           18 cycles         Return!If


;Data Manipulation
;-----------------------------------------------
p_Fill:                   8 bytes           n*21+38 cycles    Fill(,)
            n=number of bytes to fill

p_Copy:                   7 bytes           n*21+34 cycles    Copy(,,)
      n=number of bytes to copy

p_CopyRev:                7 bytes           n*21+34 cycles    Copy(,,)ʳ
      n=number of bytes to copy

p_Exchange:               15 bytes          n*62+34 cycles    Exch(,,)
      n=number of bytes to exchange

p_NibRAM:                 17 bytes          ~105 cycles
      122 cycles if even nibble, 88 cycles if odd nibble
      Call:               3 bytes           17 cycles         nib{}
```

```
p_NibApp:                  15 bytes      ~77 cycles
       84 cycles if even nibble, 71 cycles if odd nibble
       Call:               3 bytes              17 cycles           nib{}ʳ

p_NibSto:                  22 bytes      ~110 cycles
       113 cycles if even nibble, 108 cycles if odd nibble
       Call:               4 bytes              28 cycles           →nib{}

;Text
;-----------------------------------------------
;Homescreen Text
;-----------------------
p_SetCurColRow:            3 bytes              16 cycles           Output()

p_SetCur:                  8 bytes              34 cycles           Output(,)

p_DispStr:                 3 bytes              Untested            Disp
p_SetCurDispStr:           11 bytes             Untested            Output(,,)

p_DispInt:                 3 bytes              Untested            Disp ►Dec
p_SetCurDispStr:           11 bytes             Untested            Output(,,►Dec)

p_DispChar:                4 bytes              Untested            Disp ►Char
p_SetCurDispStr:           12 bytes             Untested            Output(,,►Char)

p_DispTok:                 4 bytes              Untested            Disp ►Tok
p_SetCurDispStr:           12 bytes             Untested            Output(,,►Tok)

;Homescreen App Text
;-----------------------
p_DispStrApp:              10 bytes             Untested            Disp
p_SetCurDispStrApp:        18 bytes             Untested            Output(,,)

;Graphscreen Text
;-----------------------
p_SetPenColRow:            3 bytes              16 cycles           Text()

p_SetPen:                  7 bytes              33 cycles           Text(,)

p_TextStr:                 3 bytes              Untested            Text
p_SetPenTextStr:           10 bytes             Untested            Text(,,)

p_TextInt:                 12 bytes      Untested
       Call:               3 bytes       17 cycles           Text ►Dec
       Call with SetPen:   10 bytes      49 cycles           Text(,,►Dec)

p_TextChar:                4 bytes       Untested            Text ►Char
p_SetPenTextStr:           11 bytes      Untested            Text(,,►Char)

p_TextTok:                 10 bytes      Untested            Text ►Tok
p_SetPenTextStr:           17 bytes      Untested            Text(,,►Tok)

;Graphscreen App Text
;-----------------------
p_TextStrApp:              10 bytes      Untested            Text
p_SetPenDispStrApp:        17 bytes      Untested            Text(,,)

;Text Flags
;-----------------------
p_TextSmall:               4 bytes       23 cycles           Fix 0

p_TextLarge:               4 bytes       23 cycles           Fix 1

p_TextNorm:                4 bytes       23 cycles           Fix 2

p_TextInv:                 4 bytes       23 cycles           Fix 3

p_TextScrn:                4 bytes       23 cycles           Fix 4

p_TextBuf:                 4 bytes       23 cycles           Fix 5
```

```
p_TextScroll:            4 bytes      23 cycles           Fix 6

p_TextNoScroll:          4 bytes      23 cycles           Fix 7

p_PlotToScrn:            4 bytes      23 cycles           Fix 8

p_PlotToBuff:            4 bytes      23 cycles           Fix 9


;Other
;----------------------
p_NewLine:               3 bytes      1513/~370000 cycles Disp i
      **VARIES FROM CALCULATOR TO CALCULATOR** 1513 cycles if the new line doesn't
scroll; ~370000 cycles if the new line does scroll (Speed tested in 6MHz mode on a TI-
84+SE from 2004)

p_Length:                10 bytes     n*21+32 cycles      length()
      n=length of data

p_InData:                15 bytes     See notes
      If a match is found: n*40+22 cycles, n=result; if no match is found: n*40+76
cycles, n=number of bytes of non-zero data
      Call:              5 bytes            38 cycles     inData(,)

p_Input:                 31 bytes     Untested
      Call:              3 bytes            17 cycles     input

p_ToHex:                 25 bytes     670 cycles
      Call:              3 bytes            17 cycles     ►Hex

;Screen
;--------------------------------------------
p_ClearScreen:           6 bytes            ~195000 cycles      ClrHome
            **VARIES FROM CALCULATOR TO CALCULATOR** Speed tested in 6MHz mode on a
TI-84+SE from 2004

p_ClearBuffer:           3 bytes            17296 cycles        ClrDraw

p_ClearBackBuffer:       6 bytes            17296 cycles        ClrDrawʳ

p_SaveToBuffer:          6 bytes            ~168000 cycles      StoreGDB
      **VARIES FROM CALCULATOR TO CALCULATOR** Speed tested in 6MHz mode on a TI-
84+SE from 2004

p_InvBuff:               16 bytes           28474 cycles
      Call:              3 bytes            17 cycles           DrawInv
      Call back buffer:  6 bytes            17 cycles           DrawInv ʳ
      Actually takes 27 cycles, but saves 10 cycles in the routine

p_FastCopy:              50 bytes           ~80000 cycles
      **VARIES FROM CALCULATOR TO CALCULATOR** Speed tested in 6MHz mode on a TI-
84+SE from 2004
      Call:              3 bytes            17 cycles           DispGraph
      Call any buffer:   3 bytes            17 cycles           →DispGraph
      **CURRENTLY UNAVAILABLE**

p_DrawAndClr:            51 bytes     ~80000 cycles
      Speed tested in 6MHz mode on a TI-84+SE from 2004
      Call:              3 bytes            17 cycles           DispGraphClrDraw

p_FrontToBack:           11 bytes     16153 cycles              StorePic

p_BackToFront:           11 bytes     16153 cycles              RecallPic


;Screen Shifting
;--------------------------------------------
p_ShiftLeft:             17 bytes           27542 cycles
      Call:              3 bytes            17 cycles           Horizontal -
      Call back buffer:  6 bytes            17 cycles           Horizontal -ʳ
      Actually takes 27 cycles, but saves 10 cycles in the routine
```

```
p_ShiftRight:              17 bytes            27542 cycles
      Call:                3 bytes             17 cycles           Horizontal +
      Call back buffer:    6 bytes             17 cycles           Horizontal +ʳ
      Actually takes 27 cycles, but saves 10 cycles in the routine

p_ShiftUp:                 12 bytes            15911 cycles
      Call:                3 bytes             17 cycles           Vertical -
      Call back buffer:    6 bytes             17 cycles           Vertical -ʳ
      Actually takes 27 cycles, but saves 10 cycles in the routine

p_ShiftDown:               12 bytes            15911 cycles
      Call:                3 bytes             17 cycles           Vertical +
      Call back buffer:    6 bytes             17 cycles           Vertical +ʳ
      Actually takes 27 cycles, but saves 10 cycles in the routine


;Input
;-----------------------------------------------
p_GetKey:                  6 bytes             1194 cycles         getKey

p_DKey:                    17 bytes            ~76 cycles
      If checking a specific key, 83 cycles if pressed and 71 cycles if not pressed;
if key=0 (any key), 75 cycles; if key=41 (on key), p_OnKey is used instead
      Call key≠0           6 bytes             27 cycles           getKey(15)
      Call key=0:          5 bytes             24 cycles           getKey(0)

p_DKeyExpr:                27 bytes      ~221 cycles
      m*17+n*17+110 cycles; m=key mod 8; n=key/8
      Call:                6 bytes             34 cycles           getKey()
      This command also calls p_DKey, so take into account the size and speed of
p_DKey as well

p_OnKey:                   10 bytes            58 cycles
      Call:                3 bytes             17 cycles           getKey(41)

p_Rand:                    14 bytes            88 cycles
      Call:                3 bytes             17 cycles           rand


;System
;-----------------------------------------------
p_DiagOn:                  7 bytes             ~16000 cycles       DiagnosticOn

p_DiagOff:                 7 bytes             ~16000 cycles       DiagnosticOff

p_FullSpeed:               9 bytes             41 cycles           Full

p_NormalSpeed:             3 bytes             15 cycles           Normal

p_Pause:                   7 bytes             ~n*3349-1662 cycles Pause
      n=duration argument; Pause 1800 is ~1sec at 6MHz; Pause 4500 is ~1sec at 15MHz;
Pause 0 acts like a Pause 65536

p_Contrast:                5 bytes             22 cycles           Shade()


;Linking
;-----------------------------------------------
p_PortOut:                 3 bytes             15 cycles           →Port

p_PortIn:                  7 bytes             29 cycles           Port

p_FreqOut:                 23 bytes            Untested
      Call:                5 bytes             38 cycles           Freq(,)

p_GetByte:                 42 bytes            Untested
      Call:                3 bytes             17 cycles           Get

p_SendByte:                45 bytes            Untested
      Call:                5 bytes             38 cycles           Send(
;Multiplication and division
;---------------------------------------------
p_Mul:                    22 bytes     ~898/~998 cycles
      ~m*25+798 cycles; for m, see specific call types; ~898 cycles for uniformly
distrubted byte inputs, ~998 cycles for uniformly distributed word inputs
      Call const:         6 bytes             27 cycles           *1337
      m=number of bits set in the previous multiplicand
      Call var:           7 bytes             37 cycles           *A
      m=number of bits set in the previous multiplicand
      Call expr:          5 bytes             38 cycles           *()
      m=number of bits set in the value of expr
      Call square:        5 bytes             25 cycles           ²
      m=number of bits set in the value being squared
      Call high word const:8 bytes            35 cycles           *^1337
      m=number of bits set in the previous multiplicand
      Call high word var: 9 bytes             45 cycles           *^A
      m=number of bits set in the previous multiplicand
      Call high word expr:7 bytes             46 cycles           *^()
      m=number of bits set in the value of expr

p_88Mul:                  39 bytes            ~1132 cycles
      a*19+b*19+c*18+m*25+1023; a=1 if first multiplicand is negative; b=1 if second
multiplicand is negative; c=1 if exactly one multiplicand is negative; for m, see
specific call types; n=number of bits set in the 16-bit integer part of the result;
this routine uses p_Mul, so take its size into account as well
      Call const:         6 bytes             27 cycles           **ᴇ1337
            m=number of bits set in the previous multiplicand
      Call var:           7 bytes             37 cycles           **A
      m=number of bits set in the previous multiplicand
      Call expr:          5 bytes             38 cycles           **()
      m=number of bits set in the PREVIOUS multiplicand

p_Div:                    39 bytes            ~855/~1345 cycles
      If 8-bit divisor: it's complicated... ~855 cycles; if 16-bit divisor: n*10+1194
cycles, n=number of unset bits in result, ~1345 cycles
      Call const:         6 bytes             27 cycles           /1337
      Call var:           7 bytes             37 cycles           /A
      Call expr:          6 bytes             42 cycles           /()

p_SDiv:                   36 bytes     ~1474 cycles
      Don't ask, my brain still hurts just from figuring out a decent approximation;
this routine uses p_Div, so take its size into account as well
      Call const:         6 bytes             27 cycles           //1337
      Call var:           7 bytes             37 cycles           //A
      Call expr:          6 bytes             42 cycles           //()

p_Mod:                    22 bytes     ~1319 cycles
      n*10+1166 cycles, n=number of unset bits in the 16-bit qotient of input/modulus
      Call const:         6 bytes             27 cycles           ^1337
      Call var:           7 bytes             37 cycles           ^A
      Call expr:          6 bytes             42 cycles           ^()

;Pixel Routines
;---------------------------------------------
p_Pix:                    41 bytes     ~237 cycles
      If pixel is onscreen: 178 cycles if x mod 8 = 0, n*17+177 cycles otherwise (n=x
mod 8); if pixel is vertically offscreen: 43 cycles; else if pixel is horizontally
offscreen: 63 cycles
      Call pixel on:      7 bytes             52 cycles           Pxl-On(,)
      Call pixel off:     8 bytes             56 cycles           Pxl-Off(,)
      Call pixel invert:  7 bytes             52 cycles           Pxl-Change(,)
      Call pixel test:    12 bytes            ~67 cycles          pxl-Test(,)
      66 cycles if pixel is on, 67 cycles if pixel is off


;Sprite Routines
;---------------------------------------------
p_DrawOr:                 126 bytes    829/~2000 cycles
      If fully onscreen: 829 cycles if aligned, ~2039 cycles if unaligned; if fully
or partially onscreen: ~1893 cycles; if horizontally offscreen: 82 cycles; else if
vertically offscreen: 129 cycles
```

```
        Call:                7 bytes      59 cycles          Pt-On(,,)
        Call back buffer:    13 bytes     59 cycles          Pt-On(,,)ʳ
        Actually takes 94 cycles, but saves 35 cycles in the routine
        Call const buffer:   17 bytes     80 cycles          Pt-On(,,)→L₁
        Actually takes 115 cycles, but saves 35 cycles in the routine
        Call var buffer:     17 bytes     86 cycles          Pt-On(,,)→A
        Actually takes 121 cycles, but saves 35 cycles in the routine
        Call expr buffer:    14 bytes     70 cycles          Pt-On(,,)→()
        Actually takes 105 cycles, but saves 35 cycles in the routine

p_DrawXor:                   126 bytes    829/~2000 cycles
        If fully onscreen: 829 cycles if aligned, ~2039 cycles if unaligned; if fully
or partially onscreen: ~1893 cycles; if horizontally offscreen: 82 cycles; else if
vertically offscreen: 129 cycles
        Call:                7 bytes      59 cycles          Pt-Change(,,)
        Call back buffer:    13 bytes     59 cycles          Pt-Change(,,)ʳ
        Actually takes 94 cycles, but saves 35 cycles in the routine
        Call const buffer:   17 bytes     80 cycles          Pt-Change(,,)→L₁
        Actually takes 115 cycles, but saves 35 cycles in the routine
        Call var buffer:     17 bytes     86 cycles          Pt-Change(,,)→A
        Actually takes 121 cycles, but saves 35 cycles in the routine
        Call expr buffer:    14 bytes     70 cycles          Pt-Change(,,)→()
        Actually takes 105 cycles, but saves 35 cycles in the routine

p_DrawOff:                   134 bytes    773/~2400 cycles
        If fully onscreen: 773 cycles if aligned, ~2457 cycles if unaligned; if fully
or partially onscreen: ~2255 cycles; if horizontally offscreen: 82 cycles; else if
vertically offscreen: 129 cycles
        Call:                7 bytes      59 cycles          Pt-Off(,,)
        Call back buffer:    13 bytes     59 cycles          Pt-Off(,,)ʳ
        Actually takes 94 cycles, but saves 35 cycles in the routine
        Call const buffer:   17 bytes     80 cycles          Pt-Off(,,)→L₁
        Actually takes 115 cycles, but saves 35 cycles in the routine
        Call var buffer:     17 bytes     86 cycles          Pt-Off(,,)→A
        Actually takes 121 cycles, but saves 35 cycles in the routine
        Call expr buffer:    14 bytes     70 cycles          Pt-Off(,,)→()
        Actually takes 105 cycles, but saves 35 cycles in the routine

p_EzSprite:                  7 bytes      Untested           Bitmap(,,)


;Advanced Math
;-----------------------------------------------
p_Sqrt:                      14 bytes     n*37+68 cycles
        n=result; ~444 cycles for uniformly distributed byte inputs; ~6364 cycles for
uniformly distributed word inputs
        Call:                3 bytes      17 cycles          √()

p_Sin:                       29 bytes     ~365 cycles
        Call sine:           4 bytes      21 cycles          sin()
        Call cosine:         6 bytes      28 cycles          cos()

p_Log:                       11 bytes     ~296 cycles        log()
        n*31+17 cycles, n=16-result; if input=0, n=17 (result=255); ~296 cycles for
uniformly distributed outputs

p_Exp:                       10 bytes     ~255 cycles        e^()
        n*28+45 cycles, n=input; ~255 cycles for equally distributed inputs from 0-15

;VAT manipulation
;-----------------------------------------------
p_GetCalc:                   15 bytes     Untested
        Call:                3 bytes             17 cycles          GetCalc()

p_NewVar:                    32 bytes     Untested
        Call:                4 bytes             28 cycles          GetCalc(,)

p_Unarchive:                 18 bytes     Untested
        Call:                3 bytes             17 cycles          UnArchive
```

```
p_Archive:                18 bytes     Untested
      Call:               3 bytes            17 cycles          Archive

p_DelVar:                 9 bytes      Untested                 DelVar

p_GetArc:                 59 bytes     Untested
      Call:               6 bytes            27 cycles          GetCalc(,Y_1)

p_ReadArc:                18 bytes     Untested
      Call 2 bytes:       6 bytes            30 cycles          {}^r
      This will only be used if the name of a file (Y_0-Y_9) is contained in the braces;
otherwise, the code will be treated like a normal load
      Call 1 byte:        8 bytes            37 cycles          {}
      This will only be used if the name of a file (Y_0-Y_9) is contained in the braces;
otherwise, the code will be treated like a normal load

p_CopyArc:                22 bytes     Untested
      Call:               8 bytes            52 cycles          Copy(,,)
      This will only be used if the name of a file (Y_0-Y_9) is contained in the first
argument; otherwise, the code will be treated like a normal copy


;GrayScale
;-----------------------------------------------
p_DispGS:                 64 bytes     ~66000 cycles      DispGraph^r
      **VARIES FROM CALCULATOR TO CALCULATOR** Speed tested in 6MHz mode on a TI-
84+SE from 2004

p_Disp4Lvl:               77 bytes     ~80000 cycles      DispGraph^rr
      **VARIES FROM CALCULATOR TO CALCULATOR** Speed tested in 6MHz mode on a TI-
84+SE from 2004



;Geometry Drawing
;-----------------------------------------------
p_Line:                   140 bytes    Untested
      Call:               6 bytes      50 cycles          Line(,,,)
      Call back buffer:   10 bytes     64 cycles          Line(,,,)^r

p_Box:                    114 bytes    Untested
      Faster than p_Line for horizontal or vertical lines
      Call:               6 bytes      50 cycles          Rect(,,,)
      Call back buffer    14 bytes     98 cycles          Rect(,,,)^r

p_BoxInv:                 114 bytes    Untested
      Faster than p_Line for horizontal or vertical lines
      Call:               6 bytes      50 cycles          RectI(,,,)
      Call back buffer    14 bytes     98 cycles          RectI(,,,)^r

p_Circle:                 58 bytes     Untested
      Call:               5 bytes      39 cycles          Circle(,,)

;Bit
;-----------------------------------------------
p_GetBitVar:              16 bytes     ~172 cycles        eA
      n*28+74 cycles, n=bit mod 8
p_GetBitExpr:             15 bytes     ~177 cycles        e()
      n*28+79 cycles, n=bit mod 8
p_GetBit16Var:            16 bytes     ~254 cycles        eeA
      n*24+74 cycles, n=bit mod 16
p_GetBit16Expr:           15 bytes     ~259 cycles        ee()
      n*24+79 cycles, n=bit mod 16
```

```
;Sort
;-------------------------------------------------
p_SortD:                   24 bytes     Untested
       Call:               5 bytes             38 cycles           SortD(,)


;Interrupts
;-------------------------------------------------
p_Halt:                    1 byte       4 cycles            Stop

p_FnOn:                    1 byte       4 cycles            FnOn

p_FnOff:                   1 byte       4 cycles            FnOff

p_IntOff:                  2 bytes      8 cycles            LnReg

p_IntSetup:                89 bytes     6250 cycles         fnInt(,)


;Ans
;-------------------------------------------------
p_StoreAns:                9 bytes      ~15900/~17200 cycles      →Ans
       ~15900 cycles for uniformly distributed bytes, ~17200 for uniformly distributed
words

p_RecalAns:                7 bytes      ~10100/~10300 cycles      Ans
       ~10100 cycles for uniformly distributed bytes, ~10300 cycles for uniformly
distributed values from 0-9999 (decimal)


;Specialty Drawing
;------------------------------------------------------------------------------------
p_DrawMsk:                 195 bytes    1742/~4300 cycles
       If fully onscreen: 1742 cycles if aligned, ~4397 cycles if unaligned; if fully
or partially onscreen: ~3976 cycles; if horizontally offscreen: 72 cycles; else if
vertically offscreen: 119 cycles
       Call:               7 bytes             59 cycles           Pt-Mask(,,)


;Sprite Flipping
;-------------------------------------------------
p_FlipV:                   13 bytes     322 cycles
       Call:               3 bytes             17 cycles           flipV()

p_FlipH:                   21 bytes     1891 cycles
       Call:               3 bytes             17 cycles           flipH()

p_RotC:                           20 bytes     2648 cycles
       **RETURNS INCORRECT POINTER**
       Call:               3 bytes             17 cycles           rotC()

p_RotCC:                   20 bytes     2648 cycles
       **RETURNS INCORRECT POINTER**
       Call:               3 bytes             17 cycles           rotCC()


;Floating point conversions
;-------------------------------------------------
p_FtoD:                           5 bytes             ~2000/~2200 cycles  float{}
                    ~2000 cycles for uniformly distributed bytes, ~2200 cycles for
uniformly distributed values from 0-9999 (decimal)

p_DtoF:                           14 bytes     ~3200/~4500 cycles  →float{}
       ~3200 cycles for uniformly distributed bytes, ~4500 for uniformly distributed
words
```