80x86 to Series 32000® Translation; Series 32000 Graphics Note 6

National Semiconductor
Application Note 529
Dave Rand
May 1988

1.0 INTRODUCTION

This application note discusses the conversion of Intel 8088, 8086, 80188 and 80186 (referred to here as 80x86) source assembly language to Series 32000 source code. As this is not intended to be a tutorial on Series 32000 assembly language, please see the Series 32000 Programmers Reference Manual for more information on instructions and addressing modes.

2.0 DESCRIPTION

The 80x86 model has 6 general purpose registers (AX, BX, CX, DX, SI, DI), each 16 bits wide. 4 of these registers can be further addressed as 8-bit registers (AL, AH, BL, BH, CL, CH, DL, DH). Series 32000 has 8 general purpose registers (R0–R7), each 32 bits wide. Each Series 32000 register may be accessed as an 8-, 16- or 32-bit register. Two special purpose registers on the 80x86, SP and BP, are 16-bit stack and base pointers. These are represented in Series 32000 with the SP and FP registers, each 32-bit.

The 80x86 model is capable of addressing up to 1 Megabyte of memory. Since the 16-bit register pointers are only capable of addressing 64 kbytes, 4 segment registers (CS, DS, ES, SS) are used in combination with the basic registers to point to memory. Series 32000 registers and addressing modes are all full 32-bit, and may point anywhere in the 16 Megabyte (or 4 Gigabyte, depending on processor model) addressing range.

Device ports are given their own 16-bit address on the 80x86, and there is a complement of instructions to handle input and output to these ports. Device ports on Series 32000 are memory mapped, and all instructions are available for port manipulation.

There are 6 addressing modes for data memory on the 80x86: Immediate, Direct, Direct indexed, Implied, Base relative and Stack. There are 9 addressing modes on Series 32000: Register, Immediate, Absolute, Register-relative, Memory space, External, Top-of-stack and Scaled index. Scaled index may be applied to any of the addressing modes (except scaled index) to create more addressing modes. The following figure shows the 80x86 addressing modes, and their Series 32000 counterparts.

Series 32000 assembly code reads left-to-right, meaning source is on the left, destination on the right. As you can see, most of the 80x86 addressing modes fall into the register-relative class of Series 32000. Also note that the ADDW could have been ADDD, performing a 32-bit add instead of only a 16-bit.

Series 32000 also permits memory-to-memory (two address) operation. A common operation like adding two variables is easier in Series 32000. Series 32000 has the same form for all math operations (multiply, divide, subtract), as well as all logical operators.

80x86		Series 32000
ADD AX,1234	Immediate	ADDW $1234,R0
ADD AX,LAB1	Direct	ADDW LAB1,R0
ADD AX,16[SI]	Direct Indexed	ADDW 16(R6),R0
ADD AX,[SI]	Implied	ADDW 0(R6),R0
ADD AX,[BX]	Base Relative	ADDW 0(R1),R0
ADD AX,[BX+SI]	Base Relative Implied	ADDW R1[R6:B],R0
ADD AX,12[BX+SI]	Base Relative Implied Indexed	ADDW 12(R1)[R6:B],R0
ADD AX,4[BP]	Stack (Relative)	ADDW 4(FP),R0
PUSH AX	Stack	MOVW R0,TOS

80x86	Series 32000	
MOV AL,LAB1 ADD LAB2,AL	ADDB LAB1,LAB2	8-Bit Add Operation
MOV AX,LAB3 ADD LAB4,AX	ADDW LAB3,LAB4	16-Bit Add Operation

MOV AX,LAB5L
ADD LAB6L,AX
MOV AX,LAB5H
ADDC LAB6H,AX
	ADDD LAB5,LAB6	32-Bit Add Operation

Series 32000® is a registered trademark of National Semiconductor Corporation.

Most 80x86 instructions have direct Series 32000 equivalents—with a major difference. Most 80x86 instructions affect the flags. Most Series 32000 instructions do not affect the flags in the same manner. For example, the 80x86 ADD instruction affects the Overflow, Carry, Arithmetic, Zero, Sign and Parity flags. The Series 32000 ADD instruction affects the Overflow and Carry flags. Programs that rely on side-effects of instructions which set flags must be changed in order to work correctly on Series 32000.

Table I gives a general guideline of instruction correlation between 80x86 and Series 32000. Many of the common subroutines in 80x86 may be replaced by a single instruction in Series 32000 (for example, 32-bit multiply and divide routines). Many special purpose instructions exist in Series 32000, and these instructions may help to optimize various algorithms.

3.0 IMPLEMENTATION

As an example, we will show some small 80x86 programs which we wish to convert to Series 32000. The first program reads a number of bytes from a port, waiting for status information. Below is the program in 80x86 assembly language:

```
;This program reads count bytes from port ioport, waiting for bit 7 of
;statport to be active (1) before reading each byte.
        xor     bx,bx           ;zero checksum
        mov     cx,count        ;get count of bytes
        mov     es,bufseg       ;get buffer segment
        lea     di,buffer       ;point to buffer offset
l1:     mov     dx,statport     ;get status port address
l2:     in      al,dx           ;read status port
        rcl     al,1            ;move bit 7 to carry
        jnc     l2              ;loop until status available
        mov     dx,ioport       ;point to data port
        in      al,dx           ;read port
        stosb                   ;store byte
        xor     ah,ah           ;zero high part of ax
        add     bx,ax           ;add to checksum
        loop    l1              ;loop for all bytes
        ret
```

TL/EE/9699–1

A direct translation of this program to Series 32000 using Table I, appears below. Note that this program will not work directly, due to the side effect of the rcl instruction being used.

```
#This program reads count bytes from port ioport, waiting for bit 7 of
#statport to be active (1) before reading each byte.
#
# Before optimization

        xord    r1,r1           # zero checksum
        movw    $count,r2       # get count of bytes
        addr    buffer,r5       # point to buffer
l11:    addr    statport,r3     # get status port address
l12:    movb    0(r3),r0        # read status port
        rotb    $1,r0           # move bit 7 to carry <<- does not work
        bcc     l12             # branch if carry clear
        addr    ioport,r3       # point to data port
        movb    0(r3),r0        # read port
        movb    r0,0(r5)        # store byte
        addqd   1,r5
        movzbw  r0,r0           # zero high part of ax
        addw    r0,r1           # add to checksum
        acbw    -1,r2,l11       # loop for all bytes
        ret     $0
```

TL/EE/9699–2

By using some of the special Series 32000 instructions, we can make this program much faster. The ROTB wil not work to test status, so we will replace that with a TBITB instruction. Since TBITB can directly address the port, there is no need to read the status port value at all. We will remove the read status port line, and the register load of r3. Reading the IO port as well can be done directly now, and we use a zero extension to ensure the high bits are cleared in preparation for the checksum addition. Note that it is easy to do a 32-bit checksum instead of only a 16-bit. Below is the 'optimized' code:

```
#This program reads count bytes from port ioport, waiting for bit 7 of
#statport to be active (1) before reading each byte.
#
# After optimization

        xord    r1,r1           # zero checksum
        movw    $count,r2       # get count of bytes
        addr    buffer,r5       # point to buffer
l11:
l12:    tbitb   $7,statport     # is bit 7 of status port valid?
        bfc     l12             # no, loop until it is
        movzbd  ioport,r0       # read io port
        movb    r0,0(r5)        # store in buffer
        addqd   1,r5
        addw    r0,r1           # add to checksum
        acbw    -1,r2,l11       # loop for all bytes
        ret     $0
```

TL/EE/9699–3

A second program shows, in 80x86 assembler, a method to copy and convert a string from mixed case ASCII to all upper case ASCII. This program is shown below:

```
;This program translates a null terminated ASCII string to uppercase
;
        mov     ds,buf1seg      ;point to input segment
        lea     si,buf1         ;point to input string
        mov     es,buf2seg      ;point to output segment
        lea     di,buf2         ;point to output string
        cld                     ;clear direction flag (increasing add)
l1:     lodsb                   ;get a byte
        cmp     al,'a'          ;is the char less than 'a'?
        jb      l2              ;yes, branch out
        cmp     al,'z'          ;is the char greater than 'z'?
        ja      l2              ;yes, branch out
        and     al,5fh          ;and with 5f to make uppercase
l2:     stosb                   ;store the character
        or      al,al           ;is this the last char?
        jnz     l1              ;no, loop for more
        ret                     ;yes, exit
```

TL/EE/9699–4

A direct translation to Series 32000 works fine, as is shown below:

```
#This program translates a null terminate ASCII string to uppercase
#
# Before optimization

        addr    buf1,r4         # point to input string
        addr    buf2,r5         # point to output string
ll1:    movb    0(r4),r0        # get a byte
        addqd   1,r0
        cmpb    $'a',r0         # is the char less than 'a'?
        blo     ll2             # yes, branch out
        cmpb    $'z',r0         # is the char greater than 'z'?
        bhi     ll2             # yes, branch out
```

TL/EE/9699–5

```
        andb    $0x5f,r0        # and with 5f to make uppercase
ll2:    movb    r0,0(r5)        # store the character
        addqd   1,r5
        cmpqb   0,r0            # is this the last char?
        bne     ll1             # no, loop for more
        ret     $0
```

TL/EE/9699–6

This program allows us to exploit another Series 32000 instruction, the MOVST (Move and String Translate). With a 256 byte external table, we can translate any byte to any other byte. In this example, we simply use the full range of ASCII values in the translation table, with the lower case entries containing uppercase values.

Watch for other optimization opportunities, especially with multiply and add sequences (the INDEXi instruction could be used), and possible memory to memory sequence changes. When optimizing Series 32000 code, it is important to fully utilize the Complex Instruction Set. Allow the fewest number of instructions possible to do the work. Use the advanced addressing modes where possible. Try to employ larger data types in programs (Series 32000 takes the same number of clocks to add Bytes, Words or Double words).

4.0 CONCLUSION

Series 32000 assembly language offers a much richer complement of instructions when compared to the 80x86 assembly language. Translation from 80x86 to Series 32000 is made much easier by this full instruction set.

```
#This program translates a null terminate ASCII string to uppercase
#
# After optimization

        movqd   -1,r0           # number of bytes in string max.
        addr    buf1,r1         # point to input string
        addr    buf2,r2         # point to output string
        addr    ctable,r3       # address of conversion table
        movqd   0,r4            # match on a zero
        movst   u               # move string, translate, until 0
        movqb   0,0(r2)         # move a zero to output string
        ret     $0
```

TL/EE/9699–7

TABLE I

The following is a conversion table from 80x86 mnemonics to Series 32000. Note that many of the conversions are not exact, as the 80x86 instructions may affect flags that Series 32000 instructions do not. A * marks those instructions that may be affected most by this change in flags. The i in the Series 32000 instructions refers to the size of the data to be operated on. It may be B for Byte, W for Word or D for Double. Most arithmetic instructions also support F for single-precision Floating Point, and L for double-precision Floating-Point.

80x86	Series 32000	Comments
AAA	—	Suggest changing algorithm to use ADDPi
AAD	—	Suggest changing algorithm to use ADDPi/SUBPi
AAM	—	″
AAS	—	Suggest changing algorithm to use SUBPi
ADC	ADDCi	
ADD	ADDi	
AND	ANDi	
BOUND	CHECKi	
CALL	BSR/JSR	
CBW	MOVXBW	You may directly sign-extend data while moving
CLC	BICPSRB $1	Usually not required
CLD	—	Direction encoded within string instructions
CLI	BICPSRW $0x800	Supervisor mode instruction
CMC	—	Usually not required
CMP	CMPi	
CMPS	CMPSi	Many options available
CWD	MOVXWD	You may directly sign-extend data while moving
DAA	—	Suggest changing algorithm to use ADDPi
DAS	—	Suggest changing algorithm to use SUBPi
DEC	ADDQi-1*	Watch for flag usage
DIV	DIVi	Note: Series 32000 uses signed division
ENTER	ENTER[reglist],d	Builds stack frame, saves regs, allocates stack space
ESC	—	Usually used for Floating Point-see Series 32000 FP instructions
HLT	WAIT	
IDIV	DIVi/QUOi	DIVi rounds towards -infinity, QUOi to zero
IMUL	MULi	
IN	—	Series 32000 uses memory-mapped I/O
INC	ADDQi 1*	Watch for flag usage
INS	—	Series 32000 uses memory mapped I/O
INT	SVC	Not exact conversion, but usually used to call O/S
INTO	FLAG	Trap on overflow
IRET	RETI $0	Causes Interrupt Acknowledge cycle
JA/JNBE	BHI	Unsigned comparison
JAE/JNB	BHS	Unsigned comparison
JB/JNAE	BLT	Unsigned comparison
JBE/JNA	BLS	Unsigned comparison
JCXZ	—	Use CMPQi 0, followed by BEQ
JE/JZ	BEQ	Equal comparison
JG/JNLE	BGT	Signed comparison
JGE/JNL	BGE	Signed comparison
JL/JNGE	BLT	Signed comparison
JLE/JNG	BLE	Signed comparison
JMP	BR/JUMP	
JNE/JNZ	BNE	Not Equal comparison
JNO	—	Subroutines should be used for these instructions
JNP	—	as most Series 32000 code will not need these
JNS	—	operations.
JO	—	″
JP	—	″
JPE	—	″
JPO	—	″
JS	—	″
LAHF	—	SPRB UPSR,xxx may be useful
LDS	—	Segment registers not required on Series 32000
LEA	ADDR	
LEAVE	EXIT[reglist]	Restores regs, unallocates frame and stack
LES	—	Segment registers not required
LOCK	—	SBITIi, CBITIi interlocked instructions
LODS	MOVi/ADDQD	MOV instruction followed by address increment
LOOP	ACBi-1	ACBi may use memory or register

TABLE I (Continued)

80x86	Series 32000	Comments
LOOPE	—	BEQ followed by ACBi may be used
LOOPNE	—	BNE followed by ACBi may be used
LOOPNZ	—	BNE followed by ACBi may be used
LOOPZ	—	BEQ followed by ACBi may be used
MOV	MOVi	
MOVS	MOVSi	Many options available
MUL	MULi	Series 32000 uses signed multiplication
NEG	NEGi	Two's complement
NOP	NOP	
NOT	COMi	One's complement
OR	ORi	
OUT	—	Series 32000 uses memory mapped I/O
OUTS	—	Series 32000 uses memory mapped I/O
POP	MOVi TOS,	TOS addressing mode auto increments/decrements SP
POPA	RESTORE [r0,r1 . . r7]	Restores list of registers
POPF	LPRB UPSR,TOS	User mode loads 8 bits, supervisor 16 bits of PSR
PUSH	MOVi xx,TOS	Any data may be moved to TOS
PUSHA	SAVE [r0,r1 . . r7]	Saves list of registers
PUSHF	SPRB UPSR,TOS	User mode stores 8 bits, supervisor 16 bits of PSR
RCL	ROTi*	Does not rotate through carry
RCR	ROTi*	Does not rotate through carry
REP	—	Series 32000 string instructions use 32-bit counts
RET	RET	
ROL	ROTi	
ROR	ROTi	Rotates work in both directions
SAHF	—	LPRB UPSR,xx may be useful
SAL	ASHi	Arithmetic shift
SAR	ASHi	Arithmetic shift works both directions
SBB	SUBCi	
SCAS	SKPSi	Many options available
SHL	LSHi	Logical shift
SHR	LSHi	Logical shift works both directions
STC	BISPSRB $1	
STD	—	Direction is encoded in string instructions
STI	BISPSRW $0x800	Supervisor mode instruction
STOS	MOVi/ADDQD	MOV instruction followed by address increment
SUB	SUBi	
TEST	—	TBITi may be used as a substitute
WAIT	—	
XCHG	—	MOVi x,temp; MOVi y,x; MOVi temp,y
XLAT	MOVi x[R0:b],	Scaled index addressing mode
XOR	XORi	

Lit. # 100529

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408