Assembly Language Programming for the HPC™

National Semiconductor
Application Note 510
Steve McRobert
August 1988

HOW TO WRITE SHORT, EFFICIENT, BUT UNDERSTANDABLE ASSEMBLER PROGRAMS

INTRODUCTION

One of the design objectives of the HPC family was that it should be very easy to use. With this in mind the instruction set has been designed so that it obeys a very simple set of rules. Once these rules have been learned, the programmer can write code with very little reference to instruction manuals.

The HPC is fully memory mapped. Every piece of hardware attached to an HPC core appears as a byte or a word in a linear 64K byte address space. Any data movement or arithmetic instruction can operate on any memory location and everything in the HPC has a memory location, including the accumulator. All of the I/O ports, the peripheral control registers, RAM and ROM are treated in exactly the same fashion as far as the assembly language programmer is concerned.

The HPC assembly language syntax can be explained by describing the instruction codes and the addressing modes. The instruction code tells the processor what operation it is performing, such as an add, a subtract, a multiply, a divide or a data movement instruction. The addressing mode is the way that the programmer specifies the value or values to be operated on to the microprocessor itself.

ADDRESSING MODES

Operations can be performed on any memory location. One can, for example, increment or decrement any byte or word of any memory location in the HPC. Increment and decrement are examples of single address instructions. These are instructions which have only one operand. Other examples are the bit set, bit test and bit clear instructions. These five instructions are good examples of the basic thinking behind the HPC instruction set. All of these instructions use the same four addressing modes.

Direct

The simplest addressing mode to understand is that known as direct. In this mode the address of the variable to be operated on is included as part of the sequence of bytes that comprises the entire instruction. For example, in order to perform a decrement on memory location 0F0 this value is included in the string of bytes that forms the instruction.

```
Examples:
 DECSZ   0F0.B
 INC     0F0.W
```

The increment instruction, like most other instructions with HPC, can operate on either a byte or a word. A byte access is specified by putting a B after the address of the variable, a word access by writing W.

Register Indirect

This addressing mode usually generates less bytes of code than any other. HPC has two 16-bit registers, B and X, which can be used as general purpose memory locations but also have a specific function as pointers to memory. These instructions take up very little ROM space because the address of the variable to be operated on is contained in the pointer register and the pointer register to be used is specified as part of the instruction. An instruction such as increment, using register indirect, can thus be only 1 byte long as it does not need to be followed by a byte specifying the address of the variable.

```
Examples:
 INC     [B].B ;byte increment, B pointer
 INC     [X].W ;word increment, X pointer
```

Indirect

B and X provide two 16-bit pointers to memory. Programmers will often wish to have more than two pointers in use at any one time. HPC therefore provides indirect addressing mode. In this mode a 16-bit pointer to the location to be accessed is stored in the basepage of the HPC. The instruction, therefore, is followed by a single byte which specifies the address of this 16-bit pointer. The bottom 192 bytes of RAM are on chip with the HPC and are in the so-called base page. The base page is normally used for storing frequently accessed variables as only a single byte of address is required to access a base page variable. When using indirect addressing mode, the 16-bit pointer value must always be in the base page.

```
Examples:
 DECSZ  [0].W    ;decrement a word
 INC    [0FE].B ;increment a byte
```

The base page is in the region of 0 to 0FF bytes. This area also contains the most frequently used registers such as the accumulator. The programmer can thus use indirect addressing mode with registers such as the accumulator acting as the pointer. This is an example of the simplicity of the HPC instruction set. Any operation can be performed on any HPC register simply by invoking its address in the HPC 64 kbyte addressing space.

Indexed

The last of the four basic addressing modes is indexed mode. Indexed is very similar to indirect except that an 8- or 16-bit immediate value precedes the address of the 16-bit pointer and is added to it to generate the address of the variable to be accessed. This allows a table of values to be located anywhere in memory and the pointer register need only be incremented or decremented to move through the table of values.

```
Examples:
 INC    0FF00 [4].W   ;increment a word
 DECSZ  02 [2].B      ;decrement a byte
```

Bit Operations

The bit operations of the HPC allow any bit in the memory of the HPC to be accessed. The addressing modes for these three operations, SBIT, RBIT and IFBIT, always refer to the memory location as a byte. The individual bit of the byte to be tested, using the four addressing modes already described, is actually coded into the opcode itself. This could be described as an implied addressing mode but this definition is not normally used in HPC. The way this works can be seen from the opcode map in the programmers guide of the HPC, where it can be seen that there are in fact eight opcodes shown for each of the three different bit instructions.

```
Example:
 SBIT 5, 2.B   ;set bit 5 of byte
               ;at address 2.
```

Double Register Indirect

A rule of thumb when trying to decide which addressing mode one can use with which opcode in HPC is that you can use any combination of addressing mode and opcode that is sensible. An example of this is a special addressing mode which works only for the bit instructions. This addressing mode is known as double register indirect and uses a combination of the B and X registers to index into any bit of a 64k bit string, the lower boundary of which can be located anywhere in memory.

When using this addressing mode the B register points to the lowest byte of this 8k byte string, while the most significant 13 bits of the X register point at the individual byte in the string that is being accessed. The three least significant bits of the X register point at the bit of the byte that the instruction is pointing at. By using this addressing mode, words of any length can be scanned for whether individual bits are set or cleared. This addressing mode, while unusual, fits into the scheme of things as it clearly is only of any relevance to the individual bit instructions.

```
Examples:
 SBIT X,   [B].B; Set bit
 IFBIT X,  [B] B; test bit
```

Note that the bit instructions only operate on bytes, to allow operations on words would require twice as many opcodes for no gain.

Two Address Instructions

The five instructions described so far have only one operand. There are many more instructions in the HPC instruction set which have two operands, such as arithmetic instructions, the comparison instructions and data movement instructions. The HPC instruction set allows any of these instructions to use any of the four addressing modes already described. An instruction such as multiply, for example, when written in the HPC assembler syntax as shown below shows the opcode followed by the destination operand, which is then followed by the source operand. The result of the operation in all cases except the comparison instructions winds up in the destination operand. The comparison instructions, IFEQ and IFGT do not affect the values of any memory location but, like all other two operand instructions, can operate on any two words or bytes in the HPC addressing space.

```
Examples:
 MUL A, [B].B
 MUL 0.W,2.W
```

The destination operand in HPC may be either the accumulator or a byte or word of memory accessed using the direct addressing mode. If the destination operand is the accumulator, the source operand may be addressed using direct, register indirect, indirect or indexed addressing modes as well as the familiar immediate addressing mode. The programmer can thus load the accumulator with an 8- or 16-bit immediate value which follows the opcode, multiply the accumulator with that value, divide the accumulator by that value or compare the accumulator by that value. Using the accumulator as the destination operand gives maximum flexibility in the choice of addressing mode for the source operand and also tends to produce a shorter instruction in terms of its length in bytes as the opcode does not have to include the address of the destination operand.

```
Examples:
 LD A, #37            ;load A With
                      ;immediate value.
 add 0FE.W,#  0F000 ;Add immediate to
                      ;memory.
```

Instruction Lengths

Tables are provided in the HPC users manual to allow the user to estimate the number of bytes an instruction will use and the time this instruction will take to execute. To use these tables the programmer must be aware of the name of the addressing mode he is using. This is perfectly clear for the single address instructions described at the beginning of this note but perhaps needs some explanation for two operand instructions.

For two operand instructions with the accumulator as the destination, the addressing mode is named after that used for the source operand. For example, load accumulator using a value pointed at by indirect addressing mode is referred to simply as indirect addressing mode.

Operations on Direct Memory

There are two addressing modes which allow operations to be performed directly on memory locations. If the destination operand is directly addressed memory, then the source operand may be directly addressed memory or an immediate value. These two are the only combinations of addressing modes that can be used where the destination operand is a memory location.

```
Examples:
 DIV   010.W, 0F000.W
 direct-direct mode

 DIV   0F0.B,#10
 immediate direct mode.
```

Special Symbols

Some special symbols have been allocated in the HPC cross assembler. These are A, B, K, X, PC and SP. The programmer can also define his own symbols using the equals directive of the assembler. The way that the symbols described above would be defined using the equals directive are shown below by way of example.

```
Example:
   A = 0C8.W
   B = 0CC.W
   X = 0CE.W
   K = 0CA.W
  PC = 0C6.W
  SP = 0C4.W
```

Note that these symbols cannot be redefined so the above set of definitions should never be included in a user program.

IMPLIED ADDRESSING MODES

Some of the HPC's opcodes have been shortened by using implied addressing mode. A few examples have already been shown. This section describes some more special cases. It could be said that accumulator as destination is an example of an implied addressing mode, where the address of the destination is coded into the instruction. There are some special purpose instructions which use implied addressing mode for instructions which are used very frequently. In most cases these instructions look exactly the same to the programmer as instructions using the addressing modes described earlier. For example there is a special opcode for load B with an immediate value. The programmer could do this using the immediate direct addressing mode but a special opcode has been provided to make this instruction shorter.

Load B and K is a special immediate load which loads both the B and K registers in one operation.

Carry Flag

The carry flag may be accessed using the standard bit test instructions because it can be read in the processor status word, but as carry must so often be set and tested, special instructions to do this have been included which do not require the address of the carry flag.

Multiply and Divide

Finally, the divide double and multiply instructions both have to manipulate 32-bit values. These therefore have to store an operand in two concatenated registers. The HPC instruction set cannot specify two registers with one address. Therefore these instructions default to using the X register as the high word of their 32-bit value.

The source and destination of a multiply instruction are specified as normal except that the 32-bit answer is stored in the destination operand with the 16 high bits of the answer stored in the X register. The divide double instruction basically performs the inverse of multiply, taking the 32-bit value formed by X concatenated with the destination value and dividing it by the source value. Divide double, like divide, yields a 16-bit result and a 16-bit remainder. For both divide double and divide the remainder is stored in the X register. In both cases the K register is used for intermediate value storage and is cleared as a result of this operation.

As the result of divide double can only be a 16-bit value, a full 32-bit divide is performed by following a 16-bit divide with a 32-bit divide as shown below. The example below shows how the divide instructions work together and also highlights the combinations of addressing modes that can and cannot be used with HPC.

```
                     LD          B,#11
                     DIV         HIGH.W,#10
   LOOP:             DIVD        LOW.W,#10
                     LD A, X
                     ST A, [B].B
                     DECSZ B
                     JP LOOP
```

This example shows the conversion of a 32-bit binary value in words low and high into a 10-digit BCD number in the 10 bytes starting from 1. The conversion is performed one digit at a time and the B register is used to point at the byte's location where the digit is to be stored. The first instruction of the programme therefore is to initialize the B register. The divide instruction divides word high by 10 using immediate direct addressing mode and stores the answer back in word high. The remainder is stored in the X register. The divide double instruction then divides X concatenated with word low by 10. Because X contains a remainder, the result of this division will always be a 16-bit value and can thus be stored in word low. The remainder is stored in X and is in fact the modulus and is thus the BCD digit that we have derived on this pass through the numbers.

We now wish to store the remainder into one of our BCD digit locations using register indirect mode. We need to load the value into the accumulator from X. The X register is nothing special in this application, so load A with word X is in fact an example of direct addressing mode.

Now that our BCD value is in the accumulator, we can store this in the byte location using B register indirect addressing mode.

The next instruction is decrement skip on zero. This uses direct addressing mode to decrement the B register. This instruction is an example of many in HPC which perform more than one function. As well as decrementing the memory location specified, this instruction also compares it with zero after the decrement has been performed. If the result is zero, the instruction following the decrement skip on zero instruction is skipped. That is to say it is ignored and control passes to the instruction following it. In this example the final instruction of the routine is a single byte jump back to the divide instruction. The overall loop is executed ten times in order to perform the conversion. On the final pass through the loop, B becomes zero and execution of this algorithm is terminated.

Auto Increment/Decrement Instructions

This multi-function instruction capability is best illustrated by the four special addressing modes register increment or decrement with or without conditional skip, which work only with the data movement instructions load and exchange. The load instruction in general uses any of the five two-address modes or the two combination modes to transfer data from one location to another.

The exchange instruction is similar except that the destination must always be the accumulator. Exchange not only takes the source and puts the value into the destination but also takes the value from destination and puts it into source. Clearly there is no immediate addressing mode for exchange as a destination cannot be stored into an immediate value.

When load and exchange are used with the X register as a pointer and register indirect mode, a suffix + or − can be added after the X. In this case, once the data movement operation has been performed, the X register is incremented or decremented by one or two according to whether there has been a byte or a word access respectively. A further refinement on this is provided by the load and exchange with conditional skip instructions, LDS and XS respectively. These only work with the B register as the pointer and perform two more operations rather similar to the decrement skip on zero instruction. Once the increment or decrement has been performed, the B register is compared with the K register, otherwise known as the limit register. If an increment has been performed and B is greater than K, the instruction following the movement instruction will be skipped. If a decrement is performed, the instruction is skipped if B is less than K.

An example of how these specialized instructions are used is given by the block move routine shown below;

```
         LD  X,#START
         LD  BK,#BEGIN,#END
  LOOP:  LD  A, [X+].W
         XS  A, [B+].W
         JP  LOOP
```

This routine moves a block of data from one location to another. The X register is initialized first and is used as a pointer to the first value to be moved in the source block. The B and K registers point to the first and last values respectively in the destination block. The loop itself consists of only three bytes. The first instruction loads the accumulator with the word pointed to by the X register and increments X by two. A second instruction exchanges the accumulator with the word pointed to by the B register, increments the B register by two and compares it with K. If B is greater than K, the jump instruction is skipped and this loop is terminated.

The example shows how HPC code can perform a great deal with very few instructions and use up very few bytes of code while doing so.

These auto increment/decrement instructions are the only examples where an addressing mode cannot be used for any instruction where it might make sense. It is however fairly easy to remember which addressing modes these can be used with. Auto increment/decrement can be used with the load and exchange instructions for the X register. Auto increment or decrement with conditional skip can be used with load and exchange instructions using the B register as a pointer. No other combinations are allowed.

We have not provided specific string move or search instructions but the auto increment/decrement operations provide building blocks allowing the programmer to assemble his own stock. In the block move instruction shown above, the value being moved is in the accumulator in between the load and exchange instructions. The programmer can then compare this value with anything he wishes, fill BCD to ASCII, pack BCD, unpack BCD or perform any operation he likes on a string of data.

HPC ASSEMBLY CODE

The addressing modes usable for each opcode are described in a shorthand form.

```
Example:
          ADD  MA < MA + MemI
```

In the above syntax MA means directly addressed memory or the accumulator and MemI means memory addressed using any of the four basic single-address addressing modes or an immediate value. This would be better written as shown below:

```
          A < A + MemI
or        M < M + M
or        M < M + I
```

Expanding the syntax highlights that the flexible addressing modes such as register indirect may only be used if the destination is the accumulator. It also shows that if the destination is direct memory the source may only be an immediate value or another direct memory location.

When writing assembly code the programmer writes the same mnemonic whether a memory location is a piece of RAM or ROM or an I/O port or the accumulator. In general any source or destination variable may be a byte or a word and combinations are allowed. Care must be taken when storing word into a byte location that the programmer really wishes to truncate that value to byte and throw away the upper 8 bits of the value. When loading a byte into a word location the upper 8 bits of the word location will be filled with zeros. If memory external to the HPC is used, this may be 8 or 16 bits wide. The programmer must be aware of this when writing his assembly language as HPC cannot cope with the programmer requesting a 16-bit access to 8-bit wide external memory. The HPC will not convert this to two sequential 8-bit accesses.

The only exception to this rule is that a pointer word in indirect or indexed addressing modes must always be in the base page. This is because only one byte has been allowed in the overall length of the instruction for the address of the pointer.

For all other addressing modes there is no difference in the assembly language the programmer writes between accessing a variable that is in the base page and a variable that is above address 0FF.

The programmer should be aware however that variables in the base page consume less bytes per access and the instruction will execute more quickly than non-base page variables. When studying the data sheet to see how long an instruction is, the programmer will see that the table result is different according to whether variables are base page or not. The programmer should therefore allocate base page to variables which are used most often.

EXECUTION SPEED

There are 64 bytes of RAM above the base page. These, like the base page RAM, require zero wait states to access even when the processor is running at full speed. They do however require 2 bytes of code for their addresses. These

64 bytes may best be made use of by using them as the stack area as the 16-bit stack pointer contains the full address and therefore there is no penalty in instruction length in putting the stack in this non-base page on-chip RAM.

Note that there is no difference in execution time between byte and word accesses, that is to say accesses to byte or word variables. When studying the data sheet, differences in program length and therefore in execution time will be observed according to whether the address of a directly addressed variable is a byte or a word. It is important to understand the difference between the width of the variable and the width of the address that is used to access that variable.

The cycles per instruction table is not always clear about the number of wait states applied to different variables. The HPC includes a wait state register which sets the number of wait states to be used when accessing external memory, the internal ROM, or internal registers associated with ports A and B. Wait states may be applied to these on-chip registers to allow compatibility with development tools such as the MOLE™ and HPC Designer Kit board, as when these tools are run on high clock speeds wait states must be applied for accesses to the port recreation logic. The HPC needs wait states for accessing slow external memory and when running at high clock rates.

These wait states may be applied in order that the MOLE can provide a perfect emulation of a single-chip HPC. In the MOLE the HPC is running with external memory and thus the A port and some of the B port are used for address/data and control lines respectively. The A port and part of the B port must therefore be recreated external to the HPC. In the case of the MOLE this is done using a large array of PAL®s. Because they are external to the HPC, one wait state must be applied when accessing these externally recreated ports at high clock speeds. If wait states could not be applied to these ports in a masked ROM HPC, the MOLE would not be able to provide full speed emulation. This is just one example of how the design of the HPC has been influenced by the need to emulate it 100% exactly at full speed. Apart from this no wait states are applied to any access to address locations below 200 HEX, regardless of the addressing mode used.

The HPC data sheet does not make it clear how many wait states are applied when register indirect addressing mode is used. It implies that wait states are always applied when register indirect or similar addressing modes are used, but this is not the case.

The best way to time a piece of code is to write the code and then run it through the cross assembler to generate a source plus object listing. The number of bytes generated by each instruction can then be easily read and only the cycles and accesses table need be looked up in order to calculate how long each instruction takes to execute.

Note that accesses to internal ROM are subject to at least one wait state for exactly the same reason as accesses to the A or B ports.

SUMMARY

The HPC is fully memory mapped. The I/O Ports, Peripheral Control Registers, RAM and ROM are treated exactly the same. This makes the HPC easy to program. The HPC instruction set has relatively few opcodes but allows any of these opcodes to be used with any addressing mode so as to provide an Instruction Set with great power and flexibility.

Once the contents of this note have been understood, HPC code can be written without referring to any document more lengthy than the HPC Instruction Set description in the data sheet.

Assembly Language Programming for the HPC

AN-510

Lit. # 100510

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