PART I
CHAPTER 8

EXTENDED ARITHMETIC ELEMENT

8.1 EXTENDED ARITHMETIC ELEMENT KE11-A

The Extended Arithmetic Element (EAE) (KE-11A) is an option which performs multiplication, division, multiple position shifts and normalization significantly faster than software routines. It connects directly to the UNIBUS and is programmed as a peripheral, allowing overlap between CP and EAE operations.

The KE11-A performs the following operations:

Multiply Two 16-bit numbers are multipiled to give a 32-bit product .

Examples:

000002 * 000005 = 000000-000012 $(2 * 5 = 10)_{10}$
177775 * 000007 = 177777-177753 $(-3 * 7 = -21)_{10}$

176000 * 177400 = 000004-000000 $(-2^{10} * -2^{8} = 2^{18})$
010000 * 100000 = 174000-000000 $(+12^{12} * -2^{15} = -2^{27})$

Divide A 32_{10}-bit dividend is divided by a 16_{10}-bit divisor to give a 16_{10}-bit quotient and a 16_{10}-bit remainder. The sign of the remainder is always the same as the sign of the dividend, unless the remainder is zero(i.e.–8/3 = –2REM–2 not –3 REM 1). The KE11-A indicates overflow if more than 16_{10}-bits would be needed to express the quotient (i.e. overflow if the quotient is out of the range $(2^{15})-1$ to (-2^{15}). Zero divided by zero gives overflow.

Examples:

000000-000013 / 000003 = 000003 REM 000002 $(11_{10}/3 = 3$ REM $2)$
177777-177765 / 000003 = 177775 REM 177776 $(-11_{10}/3 = -3$ REM $-2)$

000010-000000 /000020 = Overflow $2^{19}/2^{4} = 2^{15}$
000007-177777 / 000020 = 077777 REM 000017
$2^{19}-1/2^{4} = 2^{15}-1$ REM $(2^{4}-1)$
177770-000000 / 000020 = 100000 REM 000000 $(-2^{19})/2^{4} = -2^{15})$
000007-177777 / 177760 = 100001 REM 000017
$(2^{19})-1/-(2^{4}) = -((2^{15})-1)$ REM $(2^{4}-1)$

NOTE

All numbers are octal unless followed by a subscript "10" for decimal. Also, 32_{10}-bit numbers are shown in octal as two sixteen bit numbers, thus, 000001-000000 is 2^{16}.

Normalize A 32_{10}-bit number is shifted left until the two most significant bits are different. Zeros fill the empty positions on the right. A count is kept of the number of places the 32_{10}-bit number is shifted. There are three special cases:

The number is of the form 111...1100...0000 (BINARY) In this case, the number is shifted until it is 140000-000000.

The number is 177777-177777. In this case the result is 140000-000000, and the count is 30_{10}.

The number is 000000-000000. In this case the result is 000000-000000, and the count is 31_{10}.

Examples:

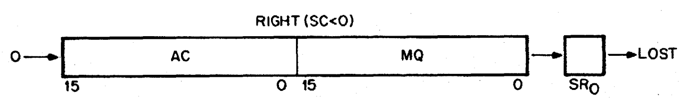
000041-170324 becomes 041741-124000 Count: 9_{10}

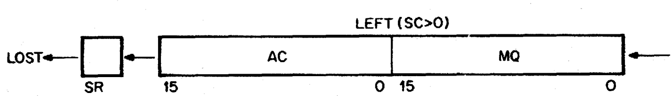
177777-174321 becomes 106420-000000 Count: 20_{10}

177740-000000 becomes 140000-000000 Count: 9_{10}

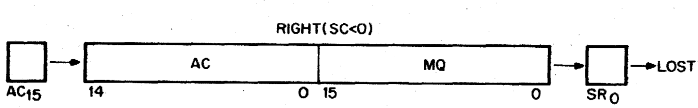
Multiple Shifts A 32_{10}-bit number is shifted either left or right the number of places specified by a count. The count is a 6-bit 2's complement number. If the count is positive, the number is shifted left; if it is negative, the number is shifted right. This allows for shifts from 31 positions left to 32 positions right. A count of zero causes no change in the number. There are two different shift operations:

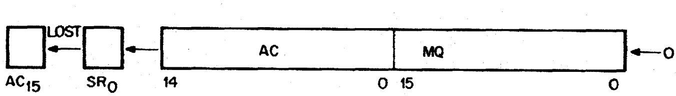
Logical Shift: Zeros always fill the vacated positions.





Arithmetic Shift: When shifting left, zeros fill the vacated positions and the most significant bit of the number is not shifted (the sign never changes). When shifting right, the most significant bit is replicated (the sign is extended).





The KE11-A indicates overflow on left shifts if the result is not the correct multiple of the original number. This occurs if the most significant bit changes on a logical shift, or if it would have changed on an arithmetic shift. No overflow is possible on right shifts.

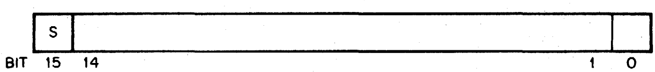
Examples:

Original Number	Count	Logical Shift	Arithmetic Shift
000777-177700	15	177770-000000	077770-000000 overflow
177525-052525	05	165252-125240	165252-125240
000777-177700	73	000017-177776	000017-177776
177525-052525	63	000007-175252	177777-175252

8.2 PROGRAMMING

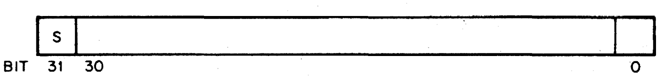
Number Formats All numbers in the KE11-A are in signed, 2's complement notation. This means that if the most significant bit of a number is zero, the number is positive and the rest of the number is the magnitude. If the most significant bit is one, it means that the number is negative and the rest of the number is the 2's complement of the magnitude. Zero is represented with all bits zero.

There are two different number formats in the KE11-A. One format uses 16_{10} bits:



This gives a range of numbers from +(215)–1 to –(215). The largest positive number is 077777 and the largest negative number is 100000. A plus one would be 000001; minus one would be 177777; and –((215)–1 would be 100001.

The other format uses 32_{10} bits:



This gives a range of numbers from (2311)–1 to –(231). The largest positive number is 077777-177777 and the largest negative number is 100000-000000. 4 The 2's complement of a number is formed by changing all 1's to 0's, all 0's to 1's, and then adding 1.

REGISTERS	ADDRESSES
Accumulator (AC)	777302
Multiplier Quotient (MQ)	777304
Step Counter (SC)	777310
Status Register (SR)	777311

Accumulator (AC) and Multiplier Quotient (MQ)

These are the two data registers in the KE-11A. Each is 16_{10}-bits. They are sometimes used together to hold one 32_{10}-bit number, in which case the MQ is the low order part of the word (bits 00-15) and the AC is the high order part (bits 16-31).

	S	AC		MQ	
BIT	31	30	16	15	0

Whenever a part of this double-word register is loaded, the sign is always extended into the higher bits that were not loaded. For example:

```
MOVB    A,MQ      ;MQ BITS 8-15 AND AC BITS 0-15 EXTENDED
MOV     A,MQ      ;AC BITS 0-15 EXTENDED
MOVB    A,MQ+1    ;AC BITS 0-15 EXTENDED
MOVB    A,AC      ;AC BITS 8-15 EXTENDED
MOV     A,AC      ;NO EXTENSION
MOVB    A,AC+1    ;NO EXTENSION
```

Thus, when loading the AC and the MQ with word operations, first the MQ and then the AC must be loaded. When using byte operations, first the low byte of the MQ, the high byte of the MQ, the low byte of the AC, and then the high byte of the AC must be loaded.

NOTE: This applies to all instructions that effect the destination not only MOVe.

On multiplication, the MQ initially contains the multiplier and the AC is ignored. After the multiply, the AC-MQ contains the 32_{10}-bit product. On division, the AC-MQ initially contains the 32_{10}-bit dividend, and after the divide, the MQ contains the quotient and the AC contains the remainder. On normalize and shifts, the AC-MQ contains the 32_{10}-bit number which is shifted.

Step Counter (SC)

The SC controls the number of steps done in all operations which the KE11-A performs. It gets loaded automatically on multiply, divide, normalize and shifting. The register is six bits long, and is at address 777310.

Status Register (SR)

The SR contains bits which give information about the last operation performed and the status of the AC and MQ. It is 8 bits long and it is at address 777311 (the high byte of the AC address).

			RO	RO	RO	RO	RO		UNUSED								
SR BITS	7	6	5	4	3	2	1	0			5	4	3	2	1	0	SC
WORD BITS	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	

RO=READ ONLY

BIT	NAME	FUNCTION
0	Carry	On shifts this bit contains the last bit shifted out of the AC-MQ.
1	AC = MQ	On multiply, divide, and normalize this 15 bit is cleared. When set, this bit means that every bit in the AC is the same as MQ bit 15, and therefore the number in the AC-MQ has only single word precision.
2	AC = MQ = 0	When set, indicates that both the MQ and AC are all zero.
3	MQ = 0	When set, indicates that the MQ is zero.
4	AC = 0	When set, indicates that the AC is zero.
5	AC = 177777	When set, indicates that the AC contains all ones.
6	NEG	On shifts, normalize, and multiply this bit is set if the AC sign bit is set. On divide, if there is no overflow, this bit is set if MQ sign bit is set. If there was overflow, this bit is set if the original dividend was negative.
7	---	This bit, in conjunction with Bit 6, is used to indicate overflow conditions. It is coded with Bit 6 as follows:

Bit 7	Bit 6	
0	0	= Positive and no overflow
0	1	= Negative and overflow
1	0	= Positive and overflow
1	1	= Negative and no overflow

The reason for coding bits 6 and 7 in this manner is so the processor condition code bits "N" and "V" can be set by a "ROLB SR" (rotate left byte) instruction. When the processor does a ROLB instruction, the old bit 6 becomes the new bit 7 and goes into condition code bit "N", and the old bit 6 exclusive-or'ed with the old bit 7 goes into condition code bit "V". Therefore, by doing a "ROLB SR" after a KE11-A operation, the "N" and "V" bits in the processor will get set, and some of the conditional branches can be used. It should be noted that the other two bits in the processor condition codes, "Z" and "C', will not be set correctly (although they will be changed) and therefore not all of the conditional branches will work.

Since it is not desirable to actually rotate the status register with the "ROLB SR", when the processor writes back the rotated SR into the KE11-A, nothing will actually change. This is done by inhibiting the SR from being written when addressed as a byte. Therefore, no instruction that attempts to write the SR as a byte will have any effect on the SR, although the KE11-A will respond normally. For example, "CLRB', "MOVB", etc. will not change the SR.

However, to allow for reentrant programming of the KE11-A, it is necessary to be able to save the SR and restore it. Therefore, when the word which contains the SR and SC is written (777310), both the SR and SC are loaded. The SC, just like the SR, however, cannot be loaded by addressing it as a byte. When reloading the registers as a word, bits 0 through 5 of the SC and bits 0, 6, and 7 of the SR are the only ones that actually change. Bits 1 to 5 of the SR always indicate the present state of the AC and MQ. Examples of reading and writing the SR and SC:

```
                        ;ASSUME THE SC = 70 AND THE SR = 140
                        ;THE COMBINED WORD IS THEN 060070
MOVB   SC,R0            ;R0 WOULD BE 000070
MOVB   SR,R0            ;R0 WOULD BE 000140
ROLB   SR               ;SR WOULD REMAIN 140, "N" AND "V"
                        BITS WOULD SET
MOVB   #-1,SC           ;SC WOULD REMAIN 70
MOVB   #-1,SR           ;SR WOULD REMAIN 140
MOV    #-1,SC           ;SC WOULD BE 77, SR WOULD BE 301.
                        ;WORD WOULD BE 140477
```

8.3 INSTRUCTIONS

Operations in the KE11-A are started by storing a number at an address. There is one address for each of the five operations that the KE11-A performs. The number must be stored as a word or as the low byte, in which case the sign is automatically extended to the high byte. Storing the number as the high byte has no effect on the KE11-A. Once an operation is initiated in the KE11-A, it will not respond to any instructions until it is finished with that operation. Thus, whenever the KE11-A is examined for a result, it will always be the correct, final answer, and never be some intermediate number. The maximum amount of time the KE11-A takes after an operation is started is 4.25 microseconds, and therefore, the most a processor can wait for a result is about 2 microseconds, due to the overlap in operation and beginning the fetch for the result.

Multiply The multiply operation is initiated by writing the 16_{10}-bit multiplicand at the multiply address. This number is then multiplied by the MQ, and a 32_{10}-bit product is left in the AC-MQ. Reading the multiply address always returns 000000.

Address:	777306
Execution Time:	4 μs
SR Bits:	0 cleared
	1, 2, 3, 4, 5 set conditionally
	6 sign of the produce (AC)
	7 no overflow possible

Divide The divide operation is initiated by writing the 16_{10}-bit divisor at the divide address. This number is then divided into the AC-MQ, and a 16_{10}-bit quotient is left in the MQ and a 16_{10}-bit remainder is left in the AC. Reading the divide address always returns 000000.

Address: 777300
Execution Time: 4.25 μs
SR Bits:
0 cleared
1, 2, 3, 4, 5 set conditionally
6 if no overflow, sign of the quotient (MQ)
if overflow, sign of the dividend (original AC sign)
7 Overflow possible

Normalize The normalize operation is initiated by writing something at the normalize address. The number written there is ignored. The operation normalizes the number in the AC-MQ. The count of the number of left shifts can be read at the normalize address, where it will be in the lower six bits. (The SR will not be in the high byte). Since the count is always a positive number, reading the normalized address as a word will get a "sign extended" value, and that number can be directly added or subtracted from an exponent.

Address: 777312
Execution Time: 0-4 μs
SR Bits:
0 cleared
1 set conditionally
2 unchanged
3, 4 set conditionally
5 cleared
6 sign of the AC
7 no overflow possible

Logical Shift The logical shift operation is initiated by writing a six bit shift count at the logical shift address. The number in the AC-MQ is then shifted right or left the number of places determined by the count. Reading the logical shift address always returns 000000.

Address: 777314
Execution Time: 0-4 μs
SR Bits:
0 Right shift: last bit shifted out of MQ(00)
Left shift: last bit shifted out of AC(15)
1, 2, 3, 4, 5 set conditionally
6 sign of the AC
7 Right shift: no overflow possible
Left shift: overflow is AC(15) changed at any point

Arithmetic Shift The arithmetic shift operation is initiated by writing a six bit shift count and the arithmetic shift address. The number in the AC-MQ is then shifted right or left the number of places determined by the count. Reading the arithmetic shift address always returns 000000.

Address: 777316
Execution Time: 0-4 μs
SR Bits:
0 Right shift: Last bit shifted out of MQ(0)
Left shift: Last bit shifted out of AC(14)
1, 2, 3, 4, 5 set conditionally
6 sign of the AC
7 Right shift: no overflow possible
Left shift: overflow if AC(15) would have changed at any point

8.4 PROGRAMMING EXAMPLES

;THE AUTO-INCREMENT AND AUTO-DECREMENT MODES OF ADDRESSING CAN BE USED TO TAKE ADVANTAGE OF THE ORDERING OF THE KE11-A ADDRESSES

```
DIV = 777300
AC = 777302
MQ = 777304
MUL = 777306
SC = 777310
SR = 777311
NOR = 777312
LSH = 777314
ASH = 777316

        ;
        MOV #MQ,R0
;SET UP R0 TO ADDRESS   OF MQ. R0 ASSUMED TO HAVE THIS ADDRESS FOR
ALL OF THESE EXAMPLES
MULTIPLY EXAMPLE
MULT:   MOV A,(0)+              ;PUT "A" INTO MQ
        MOV B,(0)               ;MULTIPLY BY "B"
        MOV -(0),C              ;PUT LOW ORDER PRODUCT IN C
        MOV -(0),D              ;PUT HIGH ORDER PRODUCT IN D
        TST (0)+                ;BUMP R0 BACK TO THE MQ
                                ;NOTE THAT IF THE PRODUCT IS KNOWN
                                TO BE LESS THAN 16 BITS, THE LAST
                                TWO LINES ABOVE CAN BE ELIMINATED.

DIVIDE EXAMPLE
DIVD:   MOV A,(0)               ;LOAD LOW ORDER DIVIDEND IN MQ
        MOV B,-(0)              ;LOAD HIGH ORDER DIVIDEND IN AC
        MOV C,-(0)              ;DIVIDE BY "C"
        TST (0)+                ;BUMP R0 BACK
        MOV (0)+,D              ;PUT REMAINDER IN "D"
        MOV (0),E               ;PUT QUOTIENT IN "E"
NORMALIZE EXAMPLE, (ASSUME AC-MQ ALREADY LOADED)
        INC @#NOR
        SUB @#NOR,R1            ;SUBTRACT COUNT FROM R1
SHIFT EXAMPLES
        MOV #3,@#LSH            ;LOGICAL SHIFT LEFT BY 3
        MOV #-5,@#ASH           ;ARITHMETIC SHIFT RIGHT BY 5
```