INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE CENTRATOM TRIESTE

H4.SMR. 403/12

FIFTH COLLEGE ON MICROPROCESSORS: TECHNOLOGY AND APPLICATIONS IN PHYSICS

2 - 27 October 1989

Assembly Language Programming

NII N. QUAYNOR
Digital Equipment Corporation, Marlboro, U.S.A.

SOFTWARE VIEW OF COMPUTER

HARDWARE ARCHITECTURE:-

(FUNCTIONS, (OPCODES)

STATE, (REGISTER, CONDITIONS)

MEMORY, (ADDRESSING,..)

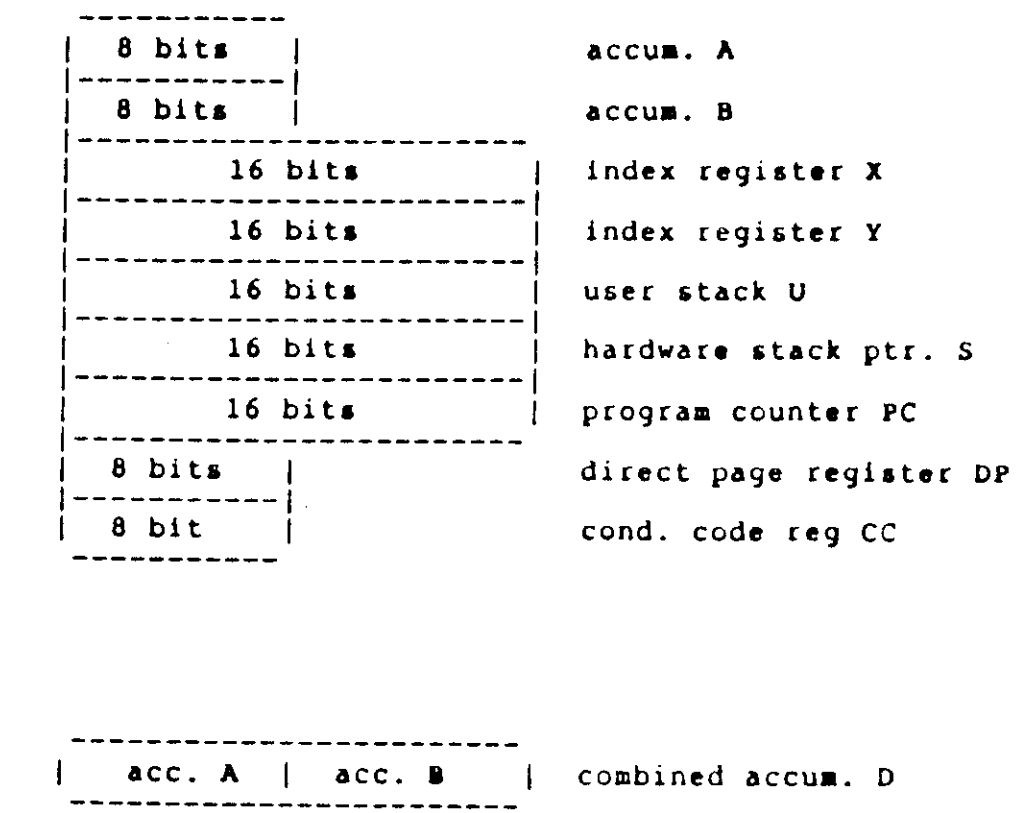
I/O (DEVICE, MAPPING, START, FINISH)
)

PROGRAMMING LANGUAGE LEVELS

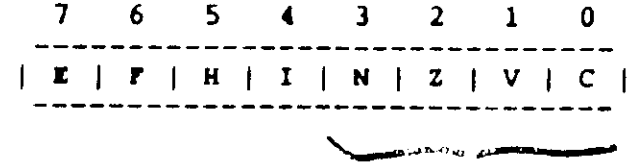
PASCAL		6809
Z:= V + W;	--> TRANSLATES	LDA V
		ADDA W
		STA Z

- o MEMORY - PROGRAM /DATA
- o PROGRAM COUNTER
- o OPCODES
- o REGISTERS
- o MEMORY ADDRESSES
- o CONDITION CODES

6809 Register Package:



MC 6809 Condition Code Register:



0 1

C:	no	with	carry or borrow
V:	no	with	magnitude overflow
Z:	not	is	zero
N:	not	is	negative
I:	enable	disable	normal interrupts
H:	no	with	half carry
F:	enable	disable	fast interrupts
E:	part	full	register stacking

FIRST PROGRAMMING PROBLEM:

MOVE CONTENTS OF MEMORY LOCATION SOURCE ($0410)
INTO MEMORY LOCATION DESTIN ($0411)

* 8-BIT DATA TRANSFER
* USE ACCUMULATOR A

MEMORY CONTENTS BEFORE EXECUTION:

LOCATION ADDRESS		SYMBOLIC NAME
040F	??	
0410	6A	SOURCE
0411	12	DESTIN

INSTRUCTIONS:

```
LDA SOURCE      * LOAD DATA FROM $0410
STA DESTIN      * TRANSFER TO $0411
```

MEMORY CONTENTS AFTER EXECUTION:

LOCATIONS:

040F	??	
0410	6A	SOURCE
0411	6A	DESTIN

SECOND PROGRAMMING PROBLEM

Add the first 15 integers Sum = 1+2+...

Constraints:

* program starts at loc $0400 - START
* result stored into loc. $0420 - RESULT

Possible programming solutions:
(expressed in a Pascal like syntax)

```
VAR
 Count   :INTEGER;        to count repetitions
 EndVal  :INTEGER;        to end repetitions
 Sum     :INTEGER;        to calculate the sum
```

--> WHILE <condition> DO <body>

```
Sum:= 0; Count:= 0;   EndVal := 15
WHILE Count < EndVal DO
BEGIN
 Count:= Count + 1;
 Sum:= Sum + Count;
END;
```

SECOND PROGRAMMING PROBLEM:

--> FOR <iterative condition> DO <body>

```
Sum:= 0;   EndVal := 15
FOR Count:= 1 to EndVal DO
   Sum:= Sum + Count;
```

--> REPEAT <body> UNTIL <condition>

```
Sum:= 0; Count:= EndVal;
REPEAT
   Sum:= Sum + Count;
   Count:= Count - 1;
UNTIL Count = 0;
```

--> LETS PROGRAM FOR 6809

DATA:

```
SUM       --> LOC.    $0420
ENDVAL    --> LOC.    $0421
COUNT     --> LOC.    $0422
```

PROGRAM:

```
          CLR     SUM       SUM:=0
          LDA     ENDVAL    COUNT:=ENDVAL
          STA     COUNT

LOOP      LDA     COUNT     REPEAT
          ADDA    SUM       SUM:=SUM+COUNT
          STA     SUM
          LDA     COUNT     COUNT:=COUNT-1
          DECA
          STA     COUNT
          BNE     LOOP      UNTIL COUNT=0
```

SECOND PROGRAMMING PROBLEM:

--> Now lets translate it: (OPCODE, LABELS)

	loc.	contents	opcode	addressing mode
	0400	7F 04 20	CLR	extended SUM
	0403	B6 04 21	LDA	extended ENDVAL
	0406	B7 04 22	STA	extended COUNT
-oop	0409	B6 04 22	LDA	extended COUNT
	040C	BB 04 20	ADDA	extended SUM
	040F	B7 04 20	STA	extended SUM
	0412	B6 04 22	LDA	extended COUNT
	0415	4A	DECA	inherent
	0416	B7 04 22	STA	extended COUNT
	419	BNE		
	41A	REL ADDR		
-->	41B			

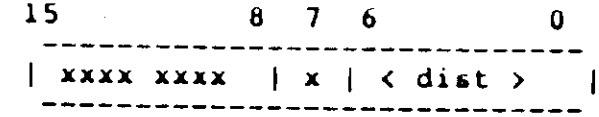
* we have to compute the relative distance

hence: $041B - $0409 --> $0012

* we branch backwards

hence: - $0012 --> $FFEE

* signed short or long branch ?



0419	26 EE	BNE	relative LOOP
041B	.....		

A VARIATION OF SECOND PROBLEM:

$$SUM_k = \sum_{i=k+1}^{k+15} i$$

DATA:

K --> $0423

PROGRAM:

	CLR	SUM
	LDA	ENDVAL
	STA	COUNT
LOOP	LDA	COUNT
	|ADDA	K|
	ADDA	SUM
	STA	SUM
	LDA	COUNT
	DECA	
	STA	COUNT
	BNE	LOOP

Assembly language elements:

Constants:

* decimal constants, (0 - 9)

  -32768 <= N <= 32767

* hexadecimal constants, (0 - 9), (A - F)

  0 <= N <= $FFFF

* octal constant (0 - 7)

  0 <= N <= @177777

* binary constant

  0 <= N <= %1111111111111111

* character constant, ASCII character prefixed with "'" (apostrophe)

Opcodes:

* mnemonics for machine instruction (CLR, DEC, etc.)
* one to one correspondence between opcodes and machine instructions
* assembler verifies opcodes and it's operands

BASIC ROUTINES - ASSEMBLY PROCESS

```
SCAN ( SYMBOL )
SEARCH ( SYMBOL, TYPE, VALUE, INDEX )
ENTER ( SYMBOL, TYPE, VALUE, INDEX )


            PASS1 PROCEDURES ( DEFINE LABELS )

PASS1_IP;
---- ---

CASE SYMBOL OF

'ORG':      LC:=NUMBER (NEXT_SYMBOL);

'RMB':      LC:=LC + NUMBER(NEXT_SYMBOL);

'FCB',
'CLRA',
'DECA',
'SWI':      LC:=LC + 1;

'BRA',
.....
'BNE';      LC:=LC + 2;

OTHERWISE   LC:=LC + 3;

ENDCASE;


PASS1_LABEL;
---- -------

SYMBOL_TABLE [ TOP ].NAME := SYMBOL
                    .TYPE := L
                    .VALUE := LC
```

2 - PASS ASSEMBLY

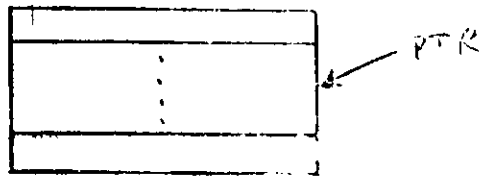
- FEW OPCODES
- ADDRESSES ARE: EXTENDED
  INHERENT
  RELATIVE (BRANCHES)
- FEW PSEUDO OPS

SYMBOL_TABLE:

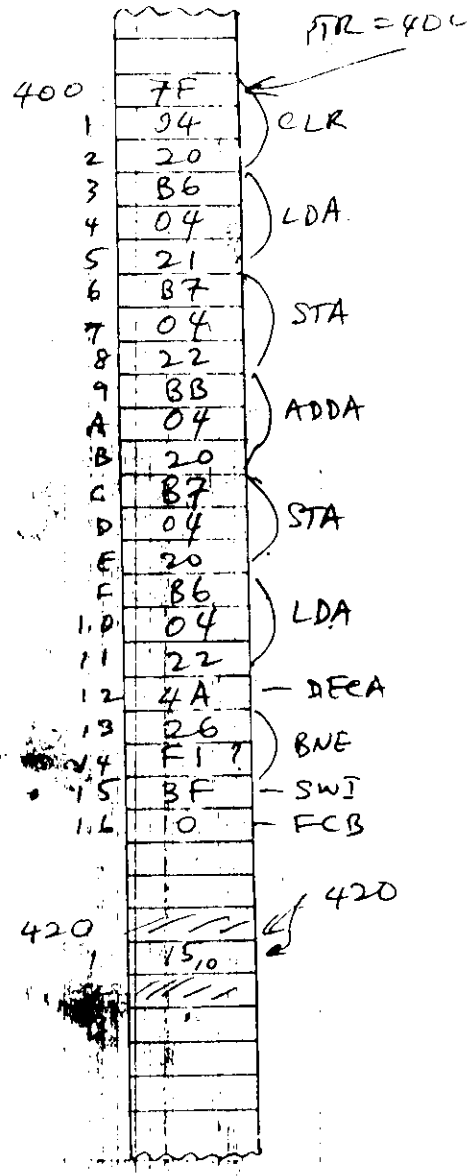
NAME	TYPE	VALUE
CLR	I	7F
LDA	I	B6
CLRA	I	4F
FCC	.	
FCB	.	
ORG	P	—
RMB		
COUNT	L	0422
SUM	L	0420
ENDVAL	L	0423
COUNT	L	0421
LOOP	L	0409

LOCATION COUNTER:

OBJECT:



OBJECT PROGRAM



4 15
-4 06
— F

0000 1111 = F
1111 0000
+ 1
— F 1

DESIGN METHODS - FLOW GRAPHS

- o MANAGE COMPLEXITY OF DESIGNS
- o STRUCTURED APPROACH TO DESIGN
- o GRAPHICAL AIDS FOR DESIGN
- o CONTROL STRUCTURES
  - SINGLE ENTRY
  - SINGLE EXIT
  - INDENT STRUCTURE
- o CORRECTNESS

+

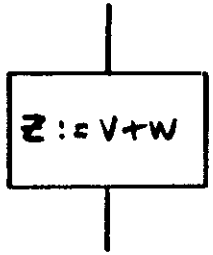
SOME ALGORITHMS

SEQUENTIAL

PASCAL

Z := V+W

NOTATION



6809

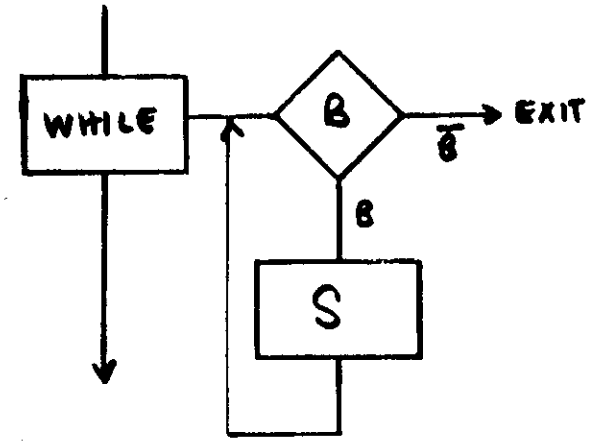
```
LDA   V
ADDA  W
STA   Z
```

ITERATION 1

PASCAL

```
WHILE B DO S;
```

NOTATION



6809

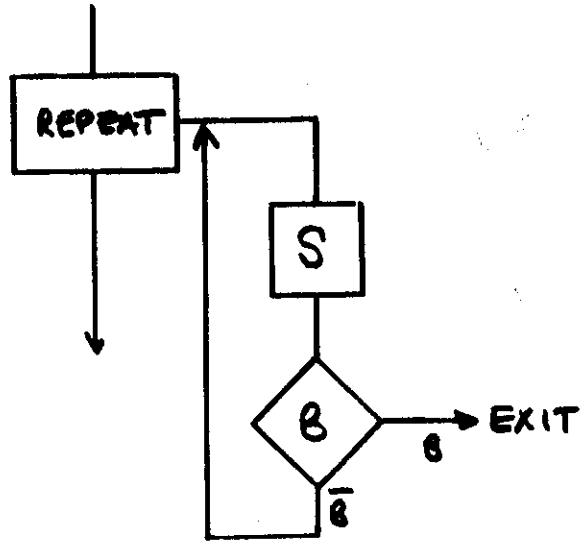
```
WHL   BRANCH ON B̄ TO EXIT
        S
      BRA  WHL
EXIT    :
        :
```


ITERATION 2

PASCAL

```
REPEAT S UNTIL B;
```

NOTATION



6809

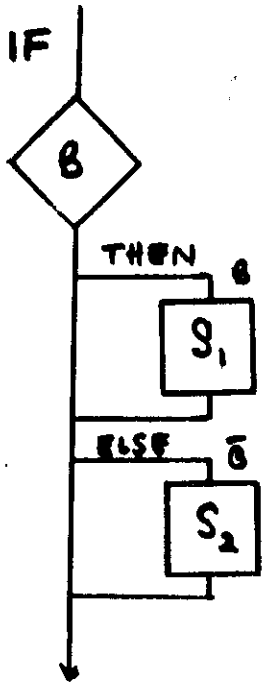
```
REP
        S
      BRANCH ON B̄ TO REP
EXIT
```

ALTERNATIVE

PASCAL

IF B THEN S_1 ELSE S_2

NOTATION

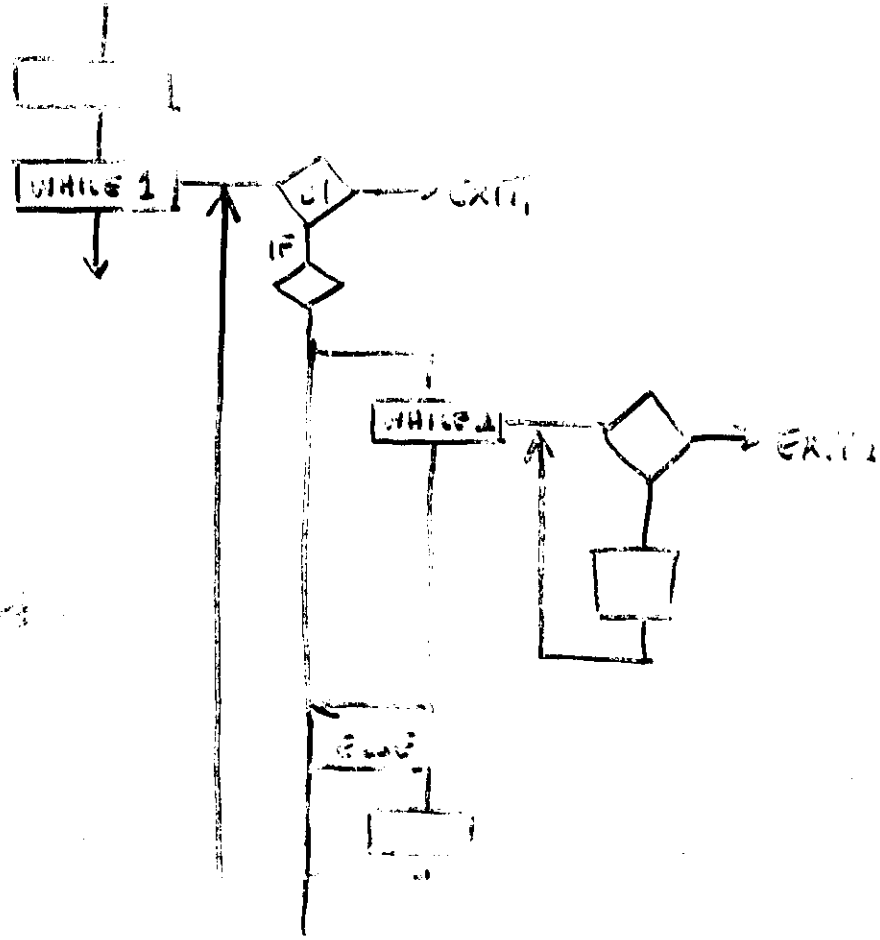


- ALSO CASE i OF $(S_1, S_2, \ldots, S_N)$

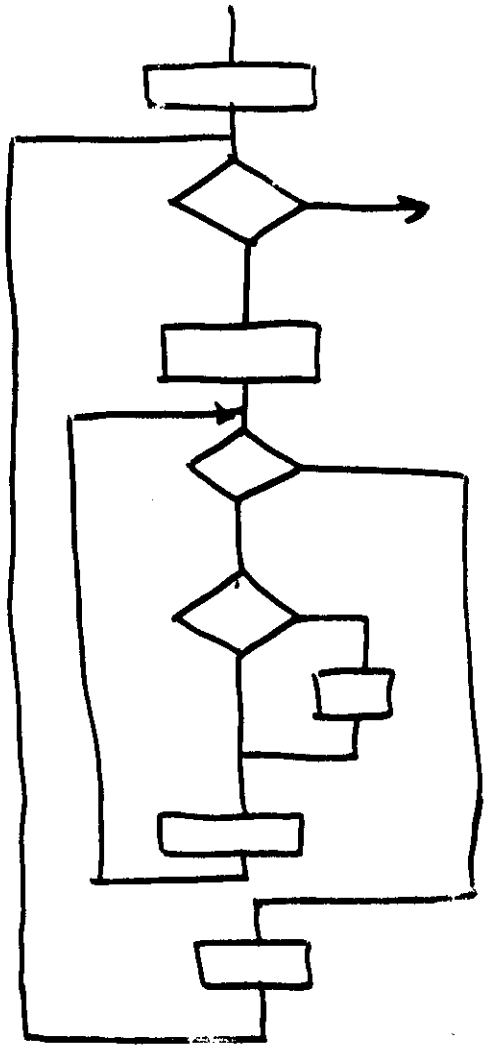
ALTERNATIVE (CONTD)

O.C.I

```
    ---
    BRANCH ON B̄ TO ELSE      IF B
      S1                     THEN S1
    BRA EXIT
ELSE                         ELSE S2
      S2
EXIT  (
      )
```



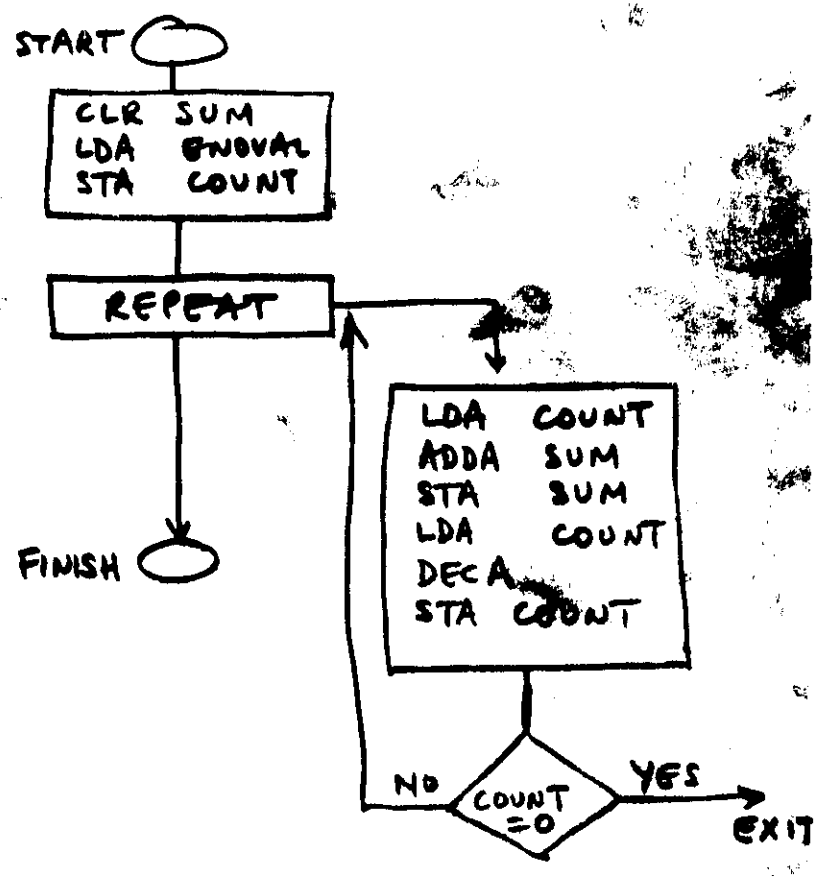
BADLY FORMED FLOW GRAPH



- MULTIPLE EXIT
- LOST STRUCTUR

FLOW DIAGRAMS - EXAMPLES

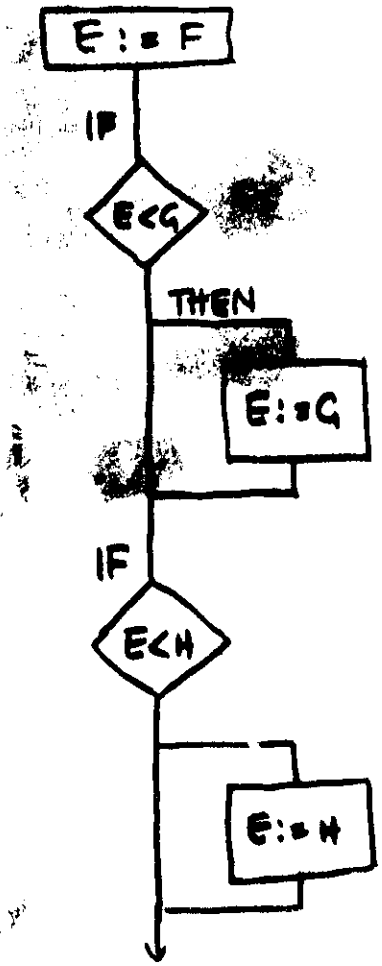
$$SUM = \sum_{i=1}^{15} i$$



EXAMPLE — IF

$E = MAX\ (F, G, H)$

UNSIGNED INTEGERS




```
        LDA   F
        CMPA  G
        BHS   EXIT1
        LDA   G
EXIT1
        CMPA  H
        BHS   EXIT2
        LDA   H
EXIT2
        STA
```

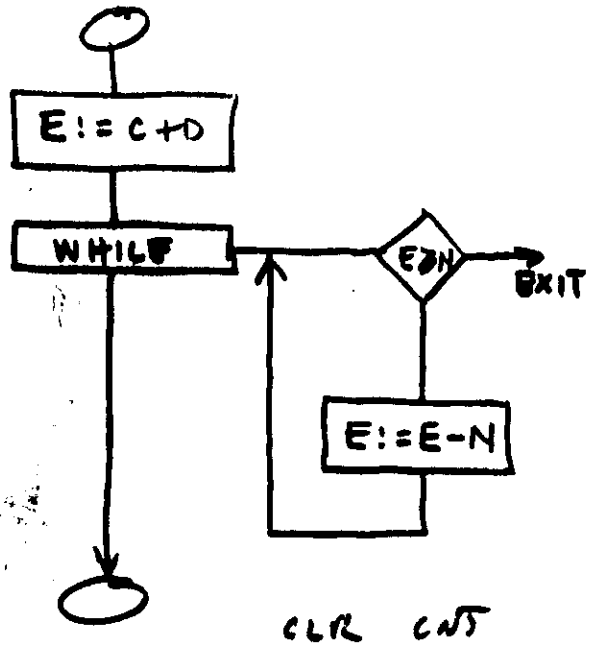
IF E<G THEN

IF E<H THEN

EXAMPLE — WHILE

E := (C+D) MOD N [UNSIGNED INT

(WITHOUT DIVISIO




```
     CLR  CNT
     LDA  C
     ADDA DD

WHL
     CMPA N
     BLO  EXIT
     INC  CNT
     SUBA N
     BRA  WHL
EXIT STA  E
```

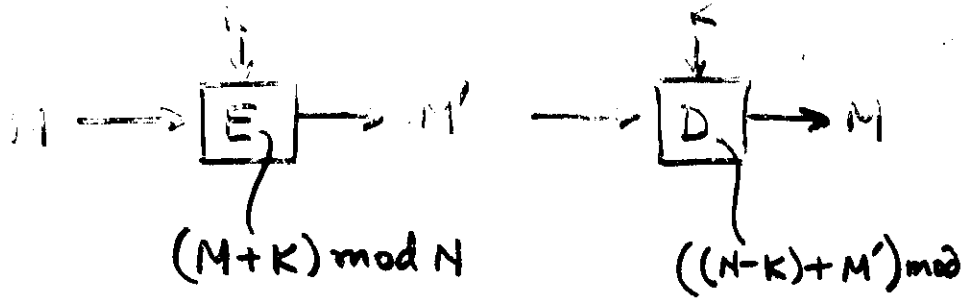
E := C+D

WHILE B DD

E := E−N

ENDWHILE

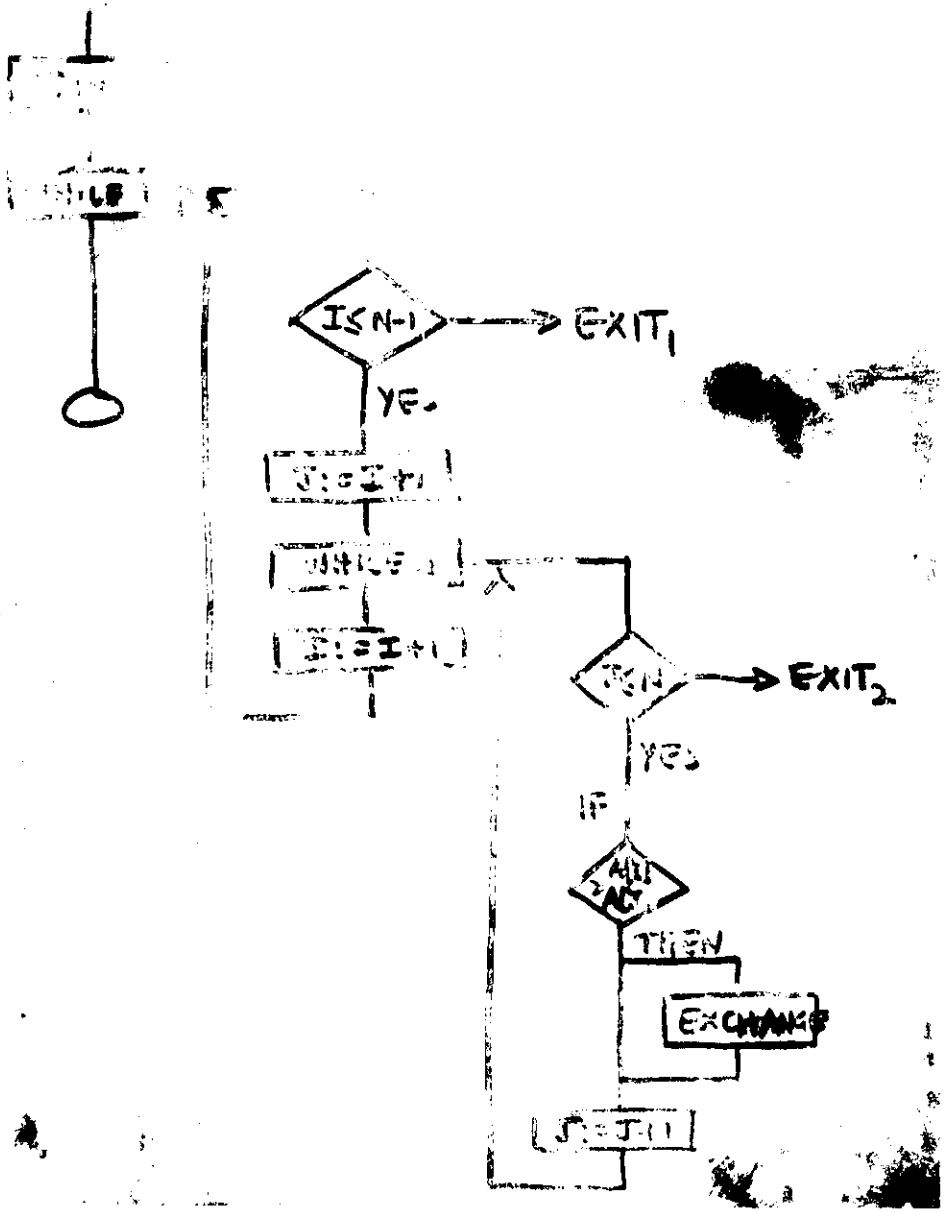
Algorithm description



EX: N=8, K=5
M=4 → M'=1 → M=4

RANDOM NUMBER GENERATION (PSEUDO)

$$R_{n+1} = (C_0 + C_1 * R_n) \bmod N$$



Third programming problem:

Now let's use indexed addressing

Structure:

--> FOR <iterative condition> DO <body>

```
FOR Index:= 1 TO 5 DO
        Bin [Index]:= 0;
```

-->

```
            NAM     Initialise an array
            ORG     $10       data area
Bin         RMB     5         reserve 5 bytes
            ORG     $200      program area
Init        CLRA              zero array "Bin"
            LDX     #0        our index
Loop        STA     Bin,X     zero next element
            LEAX    1,X       step index
            CMPX    #5        all done ?
            BNE     Loop      if not
            SWI               else --> Rosy
            FCB     0
            END     Init
```

MC 6809 effective addressing modes:

--> effective address

is the address ultimately used by the MC 6809 microprocessor for it's operation

we use <ea> as abbreviation

--> inherent addressing

operation code specifies address

```
DECA    -->     0400    4A
```

effective address for DECA is accumulator A

--> immediate addressing

data follows immediately operation code

```
LDX  #0 -->     0400    8E
                0401    00   <-- <ea>
                0402    00
```

effective address is in program counter once instruction has been decoded

MC6809 effective addressing modes:

--> extended addressing

address of data follows operation code

```
LDA  $0421      --> 0400  B6
                    0401  04
                    0402  21
```

effective address is contained in the two bytes following opeartion code

--> direct addressing

address of data is computed using:

- contents of byte after opcode
- contents of direct page register

```
STA <$0010      --> 0400  97
                    0401  10
```

assume direct page register contains 05

<ea> --> 05 10

contents of accum. A is stored into memory address 0510

MC 6809 effective addressing modes:

Indexed addressing:

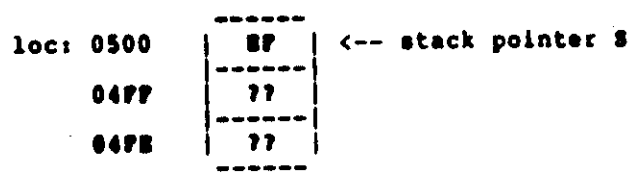
--> autodecrement / autoincrement

STD --S 0400 ED
0401 E3 (1 11 0 0011)

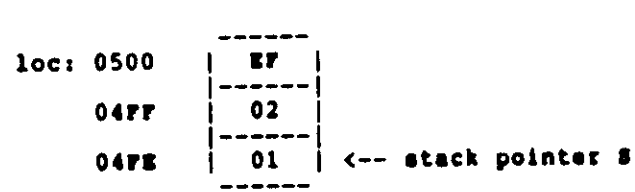
<ea> --> contents of register S decremented by two

assume: accumulator D contains: 0102
stack pointer S contains: 0500
location 0500 contains: EF

before:



after:



MC 6809 effective addressing modes:

Indexed addressing:

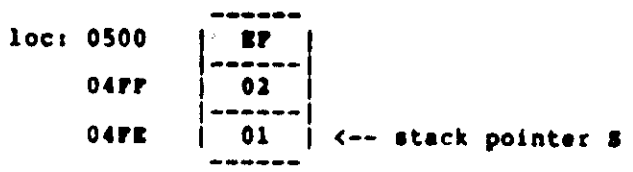
--> autodecrement / autoincrement

LDD S++ 0400 EC
0401 E1 (1 11 0 0001)

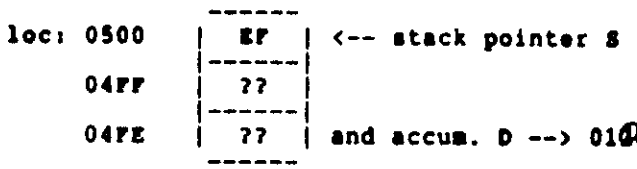
<ea> --> contents of register S incremented by two

assume: accumulator D contains: 0000
stack pointer S contains: 04FE

before:



after:



MC 6809 effective addressing modes:

--> indexed addressing

base register --> register to be used for <ea> calculation

offset --> constant value or contents of accum. to be used for <ea> calculation

post byte --> byte following opcode defines the indexed addressing mode

* the effective address is calculated using the contents of base register and the offset (if specified)

* for indirect indexed addressing mode the effective address is used to read the word at the memory location it is pointing to and use its contents as final effective address

Third programming problem:

Initialize an array to zero

Constraints:

* program starts at loc 0200

* array "Bin", 5 bytes long, starts at loc $0010

Let's do it the hard way:

```
                NAM     Initialize an array

                ORG     $10     data area

Bin             RMB     5       reserve 5 bytes

                ORG     $200    program area

Init            CLRA            zero array "Bin"
                STA     Bin+0   Bin(1)
                STA     Bin+1   Bin(2)
                STA     Bin+2   Bin(3)
                STA     Bin+3   Bin(4)
                STA     Bin+4   Bin(5)
                SWI             return to Rosy
                FCB     0

                END     Init
```

MC 6809 effective addressing modes:

Indexed adressing:

--> constant offset from base register

```
STA     Bin,X   0400    A7
                0401    88 (1 00 0 1000)
                0402    10

<ea>    -->     contents of register X
              + signed 8-bit offset in
                loc. 0402
```

or:

```
STA     Bin-1,X 0400    A7
                0401    0F (0 00 0 1111)

<ea>    -->     contents of register X
              + signed 5-bit offset in
                post byte
```

or:

```
LEAX    1,X     0400    30
                0401    01 (0 00 0 0001)

<ea>    -->     contents of register X
              + signed 5-bit offset in
                post byte
```

MC 6809 effective addressing modes:

Indexed addressing:

--> accumulator offset from base register

* the effective address is calculated by adding the signed contents of accumulator A, B or D to the contents of the base register X, Y, S, U

* this addressing mode is very useful to handle lookup tables

Example:

Assume we are reading a character from a terminal in ASCII format and have to convert it for further processing.

We use the value of the ASCII character as an index into a table containing the conversion value:

```
LDX   #Table  code table base address
LDB   CHAR    ASCII character code
LDA   B,X     corresponding code
```

Note: ASCII begins with code "00" !

MC 6809 effective addressing modes:

--> program counter relative addressing

* this address mode is implied for conditional branch instructions:

  we are branching to an instruction that is "some locations" away from where we are

  the branch instruction contains a constant signed displacement that is added to the program counter when the branch is taken

* following the same principle there is another indexed addressing mode:

  " constant offset from program counter "

MC 6809 effective addressing modes:

Indexed addressing:

--> constant offset from program counter

* the effective address is calculated by adding the signed offset to the contents of the program counter

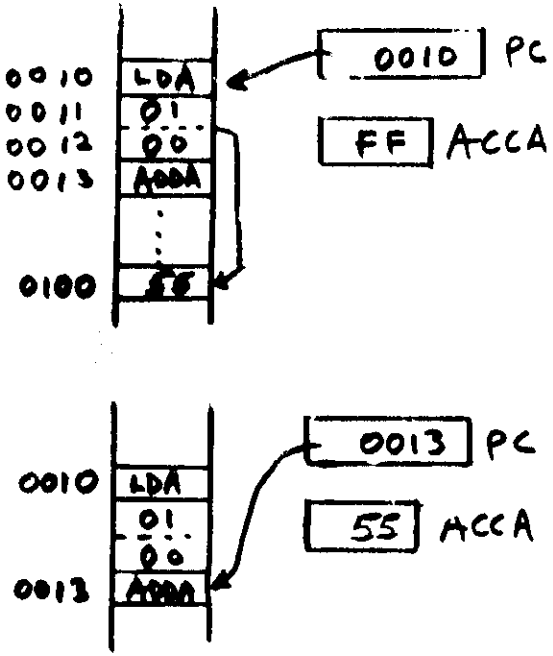
* the offset is either an 8bit or a 16-bit offset, there is no special 5-bit or accumulator offset

* the assembler will calculate the offset if we designate the address to be "program counter relative" using "pcr":

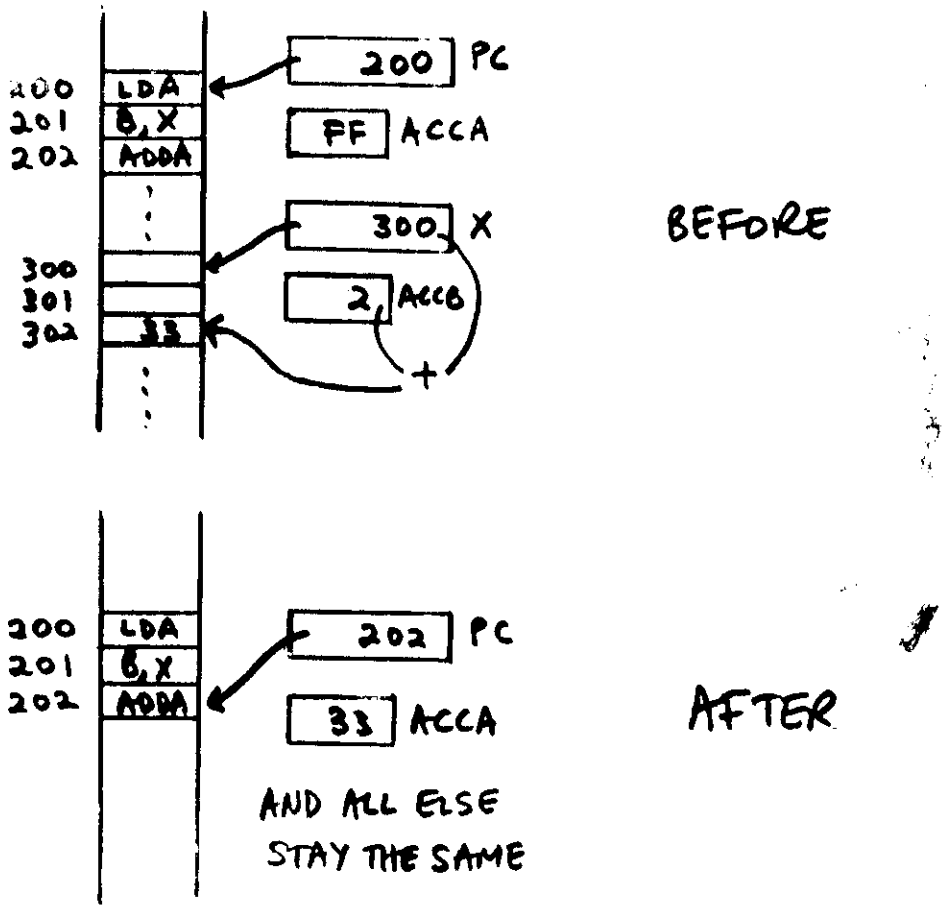
```
LEAX    L_Value,PCR
```

* this addressing mode is very useful to write position independent code.

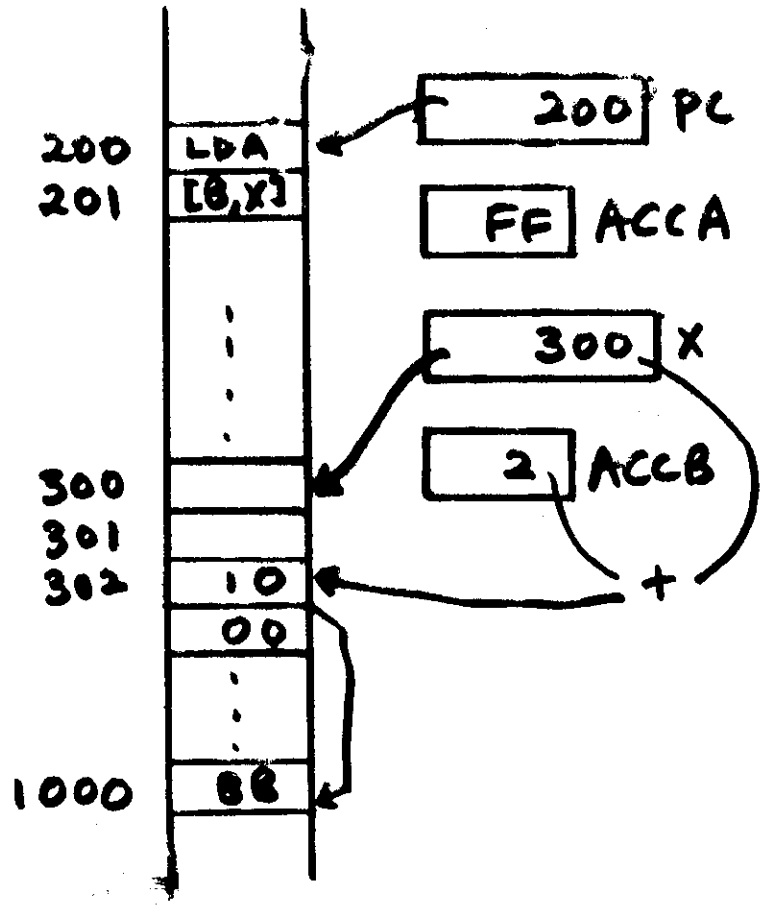
EXTENDED



REGISTER INDEXED



Indexed Indirect



ACCA = BB

APPENDIX D
PROGRAMMING AID

D.1 INTRODUCTION

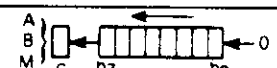
This appendix contains a compilation of data that will assist you in programming the M6809 processor. Refer to Table D-1.

Table D-1. Programming Aid

Branch Instructions

Instruction	Forms	Addressing Mode Relative OP	~	#	Description	5 H	3 N	2 Z	1 V	0 C
BCC	BCC LBCC	24 10 24	3 5(6)	2 4	Branch C = 0 Long Branch C = 0	• •	• •	• •	• •	• •
BCS	BCS LBCS	25 10 25	3 5(6)	2 4	Branch C = 1 Long Branch C = 1	• •	• •	• •	• •	• •
BEQ	BEQ LBEQ	27 10 27	3 5(6)	2 4	Branch Z = 0 Long Branch Z = 0	• •	• •	• •	• •	• •
BGE	BGE LBGE	2C 10 2C	3 5(6)	2 4	Branch ≥ Zero Long Branch ≥ Zero	• •	• •	• •	• •	• •
BGT	BGT LBGT	2E 10 2E	3 5(6)	2 4	Branch > Zero Long Branch > Zero	• •	• •	• •	• •	• •
BHI	BHI LBHI	22 10 22	3 5(6)	2 4	Branch Higher Long Branch Higher	• •	• •	• •	• •	• •
BHS	BHS LBHS	24 10 24	3 5(6)	2 4	Branch Higher or Same Long Branch Higher or Same	• •	• •	• •	• •	• •
BLE	BLE LBLE	2F 10 2F	3 5(6)	2 4	Branch ≤ Zero Long Branch ≤ Zero	• •	• •	• •	• •	• •
BLO	BLO LBLO	25 10 25	3 5(6)	2 4	Branch lower Long Branch Lower	• •	• •	• •	• •	• •
BLS	BLS LBLS	23 10 23	3 5(6)	2 4	Branch Lower or Same Long Branch Lower or Same	• •	• •	• •	• •	• •
BLT	BLT LBLT	2D 10 2D	3 5(6)	2 4	Branch < Zero Long Branch < Zero	• •	• •	• •	• •	• •
BMI	BMI LBMI	2B 10 2B	3 5(6)	2 4	Branch Minus Long Branch Minus	• •	• •	• •	• •	• •
BNE	BNE LBNE	26 10 26	3 5(6)	2 4	Branch Z ≠ 0 Long Branch Z ≠ 0	• •	• •	• •	• •	• •
BPL	BPL LBPL	2A 10 2A	3 5(6)	2 4	Branch Plus Long Branch Plus	• •	• •	• •	• •	• •
BRA	BRA LBRA	20 16	3 5	2 3	Branch Always Long Branch Always	• •	• •	• •	• •	• •
BRN	BRN LBRN	21 10 21	3 5	2 4	Branch Never Long Branch Never	• •	• •	• •	• •	• •
BSR	BSR LBSR	8D 17	7 9	2 3	Branch to Subroutine Long Branch to Subroutine	• •	• •	• •	• •	• •
BVC	BVC LBVC	28 10 28	3 5(6)	2 4	Branch V = 0 Long Branch V = 0	• •	• •	• •	• •	• •
BVS	BVS LBVS	29 10 29	3 5(6)	2 4	Branch V = 1 Long Branch V = 1	• •	• •	• •	• •	• •

Table D-1. Programming Aid (Continued)

Instruction	Forms	Addressing Modes: Immediate Op	Immediate ~	Immediate #	Direct Op	Direct ~	Direct #	Indexed Op	Indexed ~	Indexed #	Extended Op	Extended ~	Extended #	Inherent Op	Inherent ~	Inherent #	Description	5 H	3 N	2 Z	1 V	0 C
ABX														3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
ADC	ADCA	89	2	2	99	4	2	A9	4+	2+	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4+	2+	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4+	2+	BB	5	3				A + M → A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4+	2+	FB	5	3				B + M → B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6+	2+	F3	7	3				D + M:M + 1 → D	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4+	2+	B4	5	3				A ∧ M → A	•	↕	↕	0	•
	ANDB	C4	2	2	D4	4	2	E4	4+	2+	F4	5	3				B ∧ M → B	•	↕	↕	0	•
	ANDCC	1C	3	2													CC ∧ IMM → CC					↕7
ASL	ASLA													48	2	1		↕8	↕	↕	↕	↕
	ASLB													58	2	1		↕8	↕	↕	↕	↕
	ASL				08	6	2	68	6+	2+	78	7	3					↕8	↕	↕	↕	↕
ASR	ASRA													47	2	1		↕8	↕	↕	•	↕
	ASR													57	2	1		↕8	↕	↕	•	↕
	ASR				07	6	2	67	6+	2+	77	7	3					↕8	↕	↕	•	↕
BIT	BITA	85	2	2	95	4	2	A5	4+	2+	B5	5	3				Bit Test A (M ∧ A)	•	↕	↕	0	•
	BITB	C5	2	2	D5	4	2	E5	4+	2+	F5	5	3				Bit Test B (M ∧ B)	•	↕	↕	0	•
CLR	CLRA													4F	2	1	0 → A	•	0	1	0	0
	CLRB													5F	2	1	0 → B	•	0	1	0	0
	CLR				0F	6	2	6F	6+	2+	7F	7	3				0 → M	•	0	1	0	0
CMP	CMPA	81	2	2	91	4	2	A1	4+	2+	B1	5	3				Compare M from A	↕8	↕	↕	↕	↕
	CMPB	C1	2	2	D1	4	2	E1	4+	2+	F1	5	3				Compare M from B	↕8	↕	↕	↕	↕
	CMPD	10 83	5	4	10 93	7	3	10 A3	7+	3+	10 B3	8	4				Compare M:M + 1 from D	•	↕	↕	↕	↕
	CMPS	11 8C	5	4	11 9C	7	3	11 AC	7+	3+	11 BC	8	4				Compare M:M + 1 from S	•	↕	↕	↕	↕
	CMPU	11 83	5	4	11 93	7	3	11 A3	7+	3+	11 B3	8	4				Compare M:M + 1 from U	•	↕	↕	↕	↕
	CMPX	8C	4	3	9C	6	2	AC	6+	2+	BC	7	3				Compare M:M + 1 from X	•	↕	↕	↕	↕
	CMPY	10 8C	5	4	10 9C	7	3	10 AC	7+	3+	10 BC	8	4				Compare M:M + 1 from Y	•	↕	↕	↕	↕
COM	COMA													43	2	1	$\overline{A} \rightarrow A$	•	↕	↕	0	1
	COMB													53	2	1	$\overline{B} \rightarrow B$	•	↕	↕	0	1
	COM				03	6	2	63	6+	2+	73	7	3				$\overline{M} \rightarrow M$	•	↕	↕	0	1
CWAI		3C	≥20	2													CC ∧ IMM → CC, Wait for Interrupt					↕7
DAA														19	2	1	Decimal Adjust A	•	↕	↕	0	↕
DEC	DECA													4A	2	1	A − 1 → A	•	↕	↕	↕	•
	DECB													5A	2	1	B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	6A	6+	2+	7A	7	3				M − 1 → M	•	↕	↕	↕	•
EOR	EORA	88	2	2	98	4	2	A8	4+	2+	B8	5	3				A ⊻ M → A	•	↕	↕	0	•
	EORB	C8	2	2	D8	4	2	E8	4+	2+	F8	5	3				B ⊻ M → B	•	↕	↕	0	•
EXG	R1, R2	1E	8	2													R1 → R2[2]	•	•	•	•	•
INC	INCA													4C	2	1	A + 1 → A	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	6C	6+	2+	7C	7	3				M + 1 → M	•	↕	↕	↕	•
JMP					0E	3	2	6E	3+	2+	7E	4	3				EA[3] → PC	•	•	•	•	•
JSR					9D	7	2	AD	7+	2+	BD	8	3				Jump to Subroutine	•	•	•	•	•
LD	LDA	86	2	2	96	4	2	A6	4+	2+	B6	5	3				M → A	•	↕	↕	0	•
	LDB	C6	2	2	D6	4	2	E6	4+	2+	F6	5	3				M → B	•	↕	↕	0	•
	LDD	CC	3	3	DC	5	2	EC	5+	2+	FC	6	3				M:M + 1 → D	•	↕	↕	0	•
	LDS	10 CE	4	4	10 DE	6	3	10 EE	6+	3+	10 FE	7	4				M:M + 1 → S	•	↕	↕	0	•
	LDU	CE	3	3	DE	5	2	EE	5+	2+	FE	6	3				M:M + 1 → U	•	↕	↕	0	•
	LDX	8E	3	3	9E	5	2	AE	5+	2+	BE	6	3				M:M + 1 → X	•	↕	↕	0	•
	LDY	10 8E	4	4	10 9E	6	3	10 AE	6+	3+	10 BE	7	4				M:M + 1 → Y	•	↕	↕	0	•
LEA	LEAS							32	4+	2+							EA[3] → S	•	•	•	•	•
	LEAU							33	4+	2+							EA[3] → U	•	•	•	•	•
	LEAX							30	4+	2+							EA[3] → X	•	•	↕	•	•
	LEAY							31	4+	2+							EA[3] → Y	•	•	↕	•	•

Legend:

OP	Operation Code (Hexadecimal)	$\overline{M}$	Complement of M	↕	Test and set if true, cleared otherwise
~	Number of MPU Cycles	→	Transfer Into	•	Not Affected
#	Number of Program Bytes	H	Half-carry (from bit 3)	CC	Condition Code Register
+	Arithmetic Plus	N	Negative (sign bit)	:	Concatenation
−	Arithmetic Minus	Z	Zero (Reset)	V	Logical or
•	Multiply	V	Overflow, 2's complement	∧	Logical and
		C	Carry from ALU	⊻	Logical Exclusive or

Table D-1. Programming Aid (Continued)

Instruction	Forms	Addressing Modes: Immediate Op	Immediate ~	Immediate #	Direct Op	Direct ~	Direct #	Indexed[1] Op	Indexed ~	Indexed #	Extended Op	Extended ~	Extended #	Inherent Op	Inherent ~	Inherent #	Description	5 H	3 N	2 Z	1 V	0 C
LSL	LSLA													48	2	1	A, B, M: C b7 b0 ← 0	•	↕	↕	↕	↕
	LSLB													58	2	1		•	↕	↕	↕	↕
	LSL				08	6	2	68	6+	2+	78	7	3					•	↕	↕	↕	↕
LSR	LSRA													44	2	1	A, B, M: 0 → b7 b0 C	•	0	↕	•	↕
	LSRB													54	2	1		•	0	↕	•	↕
	LSR				04	6	2	64	6+	2+	74		3					•	0	↕	•	↕
MUL														3D	11	1	A x B — D (unsigned)	•	•	↕	•	9
NEG	NEGA													40	2	1	$\overline{A}$ + 1 — A	8	↕	↕	↕	↕
	NEGB													50	2	1	$\overline{B}$ + 1 — B	8	↕	↕	↕	↕
	NEG				00	6	2	60	6+	2+	70	7	3				$\overline{M}$ + 1 — M	8	↕	↕	↕	↕
NOP														12	2	1	No Operation	•	•	•	•	•
OR	ORA	8A	2	2	9A	4	2	AA	4+	2+	BA	5	3				A v M — A	•	↕	↕	0	•
	ORB	CA	2	2	DA	4	2	EA	+	2+	FA	5	3				B v M — B	•	↕	↕	0	•
	ORCC	1A	3	2													CC v IMM — CC	↕	↕	↕	↕	↕
PSH	PSHS	34	5+[4]	2													Push Registers on S Stack	•	•	•	•	•
	PSHU	36	5+[4]	2													Push Registers on U Stack	•	•	•	•	•
PUL	PULS	35	5+[4]	2													Pull Registers from S Stack	•	•	•	•	•
	PULU	37	5+[4]	2													Pull Registers from U Stack	•	•	•	•	•
ROL	ROLA													49	2	1	A, B, M: C b7 b0	•	↕	↕	↕	↕
	ROLB													59	2	1		•	↕	↕	↕	↕
	ROL				09	6	2	69	6+	2+	79	7	3					•	↕	↕	↕	↕
ROR	RORA													46	2	1	A, B, M: C b7 b0	•	↕	↕	•	↕
	RORB													56	2	1		•	↕	↕	•	↕
	ROR				06	6	2	66	6+	2+	76	7	3					•	↕	↕	•	↕
RTI														3B	6/15	1	Return From Interrupt	↕	↕	↕	↕	7
RTS														39	5	1	Return from Subroutine	•	•	•	•	•
SBC	SBCA	82	2	2	92	4	2	A2	4+	2+	B2	5	3				A - M - C — A	8	↕	↕	↕	↕
	SBCB	C2	2	2	D2	4	2	E2	4+	2+	F2	5	3				B - M - C — B	8	↕	↕	↕	↕
SEX														1D	2	1	Sign Extend B into A	•	↕	↕	0	•
ST	STA				97	4	2	A7	4+	2+	B7	5	3				A — M	•	↕	↕	0	•
	STB				D7	4	2	E7	4+	2+	F7	5	3				B — M	•	↕	↕	0	•
	STD				DD	5	2	ED	5+	2+	FD	6	3				D — M M + 1	•	↕	↕	0	•
	STS				10	6	3	10	6+	3+	10	7	4				S — M M + 1	•	↕	↕	0	•
					DF			EF			FF											
	STU				DF	5	2	EF	5+	2+	FF	6	3				U — M M + 1	•	↕	↕	0	•
	STX				9F	5	2	AF	5+	2+	BF	6	3				X — M M + 1	•	↕	↕	0	•
	STY				10	6	3	10			10	7	4				Y — M M + 1	•	↕	↕	0	•
					9F			AF	6+	3+	BF											
SUB	SUBA	80	2	2	90	4	2	A0	4+	2+	B0	5	3				A - M — A	8	↕	↕	↕	↕
	SUBB	C0	2	2	D0	4	2	E0	4+	2+	F0	5	3				B - M — B	8	↕	↕	↕	↕
	SUBD	83	4	3	93	6	2	A3	6+	2+	B3	7	3				D - M M + 1 — D	•	↕	↕	↕	↕
SWI	SWI[6]													3F	19	1	Software Interrupt 1	•	•	•	•	•
	SWI2[6]													10 3F	20	2	Software Interrupt 2	•	•	•	•	•
	SWI3[6]													11 3F	20	1	Software Interrupt 3	•	•	•	•	•
SYNC														13	≥4	1	Synchronize to Interrupt	•	•	•	•	•
TFR	R1 R2	1F	6	2													R1 — R2[2]	•	•	•	•	•
TST	TSTA													4D	2	1	Test A	•	↕	↕	0	•
	TSTB													5D	2	1	Test B	•	↕	↕	0	•
	TST				0D	6	2	6D	6+	2+	7D	7	3				Test M	•	↕	↕	0	•

Notes:

1. This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, in Appendix F.
2. R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers.
   The 8 bit registers are: A, B, CC, DP
   The 16 bit registers are: X, Y, U, S, D, PC
3. EA is the effective address
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each byte pushed or pulled.
5. 5(6) means 5 cycles if branch not taken, 6 cycles if taken (Branch instructions)
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a direct result of the instruction.
8. Value of half-carry flag is undefined.
9. Special Case — Carry set if b7 is SET.

Table D-1. Programming Aid (Continued)

SIMPLE BRANCHES

	OP	~	#
BRA	20	3	2
LBRA	16	5	3
BRN	21	3	2
LBRN	1021	5	4
BSR	8D	7	2
LBSR	17	9	3

SIMPLE CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
N = 1	BMI	2B	BPL	2A
Z = 1	BEQ	27	BNE	26
V = 1	BVS	29	BVC	28
C = 1	BCS	25	BCC	24

SIGNED CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
r > m	BGT	2E	BLE	2F
r ≥ m	BGE	2C	BLT	2D
r = m	BEQ	27	BNE	26
r ≤ m	BLE	2F	BGT	2E
r < m	BLT	2D	BGE	2C

UNSIGNED CONDITIONAL BRANCHES (Notes 1-4)

Test	True	OP	False	OP
r > m	BHI	22	BLS	23
r ≥ m	BHS	24	BLO	25
r = m	BEQ	27	BNE	26
r ≤ m	BLS	23	BHI	22
r < m	BLO	25	BHS	24

Notes:

1. All conditional branches have both short and long variations.
2. All short branches are 2 bytes and require 3 cycles.
3. All conditional long branches are formed by prefixing the short branch opcode with $10 and using a 16-bit destination offset.
4. All conditional long branches require 4 bytes and 6 cycles if the branch is taken or 5 cycles if the branch is not taken.