Microprocessor Systems

Logical Operations

- AND , OR, exclusive OR: instructions are used to perform the boolean logical operations AND, OR, exclusive OR. Accumulators A and B, can be logically ANDed, ORed, or exclusive-ORed with a register, memory or immediate value.

AND A, $25	ACC A ←	ACC A .AND. $25
OR A, B	ACC A ←	ACC A .OR. B
XOR A, <$1000>	ACC A ←	ACC A .XOR. <1000>

Logical Operations

- One’s complement: Accumulators, registers and specified memory locations can be logically complemented (bit-wise).

COM A	ACC A	←	one’s complement of ACCA
COM C	C	←	one’s complement of C
COM <$1000>	<$1000>	←	one’s complement of <$1000>

- Two’s complement : Contents of an accumulator, register or memory location are converted to two’s complement.

NEG A	ACC A	←	two’s complement of ACC A
NEG C	C	←	two’s complement of C
NEG <$1000>	<$1000>	←	two’s complement of <$1000>

Logical Operations

- Clear: Clear writes zeros into the destination operand.

CLR A	ACC A $\longleftarrow$ 0
CLR <$1000>	<$1000> $\leftarrow$ 0
CLR C	C $\longleftarrow$ 0 (Carry flag is set to 0)

In the absence of clear:

- XOR with the same content.

- Set: Flags in the CCR are set to one

SET C	set carry flag to 1
SET H	set half carry flag to one
SET Z	set zero flag to one
SET V	set overflow flag to one
SET N	set negative flag to one.

Logical Operations

- Decimal Adjustment: Translate a binary number in an accumulator to BCD.

  DAA A

  DAA B

- Increment: Increment instructions are used to increment a variable by one.

  INC A   ACC A ← ACC A + 1

  INC <$1000>   <$1000>   <$1000> + 1

- Decrement: Decrement instructions are used to decrement a variable by one.

  DEC A   ACC A   ACC A - 1

  DEC <$1000>   <$1000>   <$1000> - 1

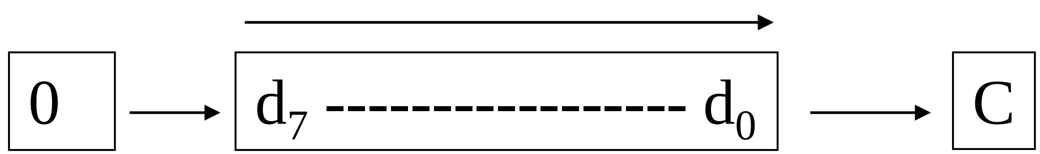
Shift and Rotate Instructions

- All the shift and rotate instructions involve the carry bit in the CCR in addition to the 8 bit operand in the instruction, which permits easy extension to multiple-word operands.

- Also, by setting or clearing the carry bit before a shift or rotate instruction, the programmer can easily control what will be shifted into the opened end of an operand.

Shift Instructions

- Logical Shift Right: Shift instructions can operate on accumulators, registers, or on a memory location. They are used for unsigned numbers.



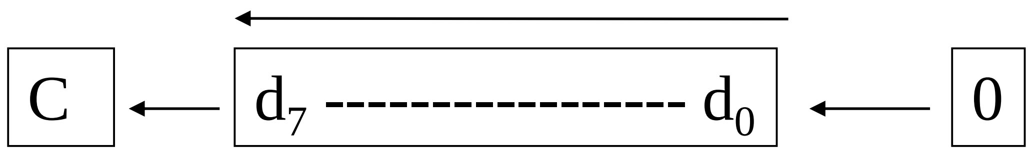
LSR A

LSR B

LSR <$1000>

Shift Instructions

- Logical Shift left



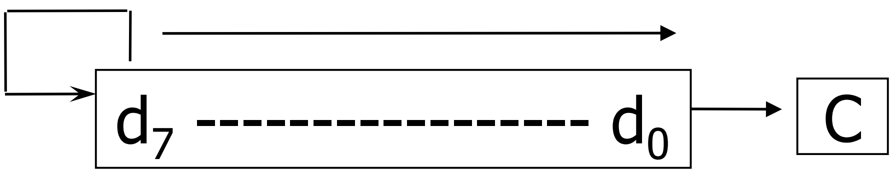
LSL A

LSL B

LSL <$1000>

Shift Instructions

- Arithmetic shift right**:** The arithmetic shift right instruction maintains the original value of the MSB of the operand.



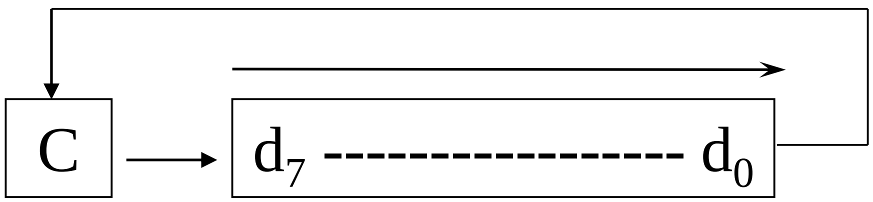
ASR A

ASR B

ASR <$1000>

Rotate Instructions

- Circular Shift Right: It rotates the contents of accumulators or a memeory location to the right by 1-bit position.



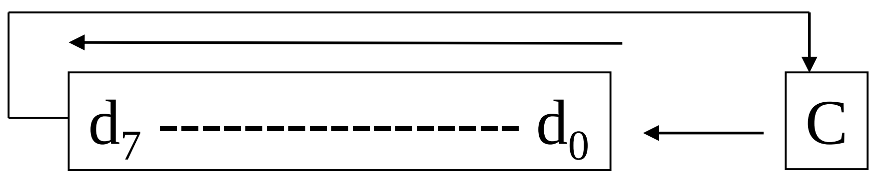
ROR A

ROR B

ROR <$1000>

Rotate Instructions

- Circular Shift Left



- ROL A
- ROL B
- ROL <$1000>

Controlling the Flow of Execution

- So far we’ve seen only programs that execute instructions in strict sequence until a SWI is reached.

- Most programs, however, must sometimes **jump** or **branch** to execute something other than the next instruction.

- The jump or branch can be **unconditional**, perhaps to skip from the last location of one block of installed memory to the first location of the next, or it can be **conditional**.

Unconditional Branches

- Branch Always: Branch-always instructions transfer control unconditionally to a location specified using **relative addressing**.

  BRA Step

- Jump: The program branches to the **memory location specified in the instruction**.

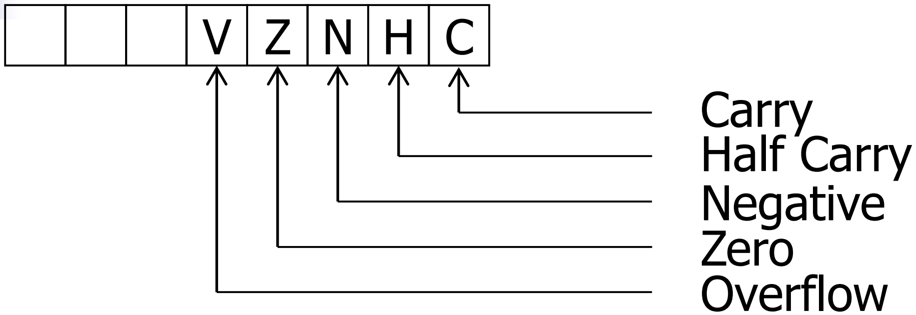
  JMP <ADDRESS>

Conditional Branches

- Conditional branching is especially important, as it underlies the implementation of such familiar and important high-level language constructs as **if-then-else**, **do-while**, **repeat-until**, and **for**.

- There are actually two distinct groups, which operate quite differently:
  - <u>Simple conditional</u> branches base their decisions on a single bit in the condition code register (CCR).
  - <u>Comparison</u> branches can base their decisions on more than one CCR bit, and offer the ability to condition branches on higher-level criteria.

The Condition Code Register



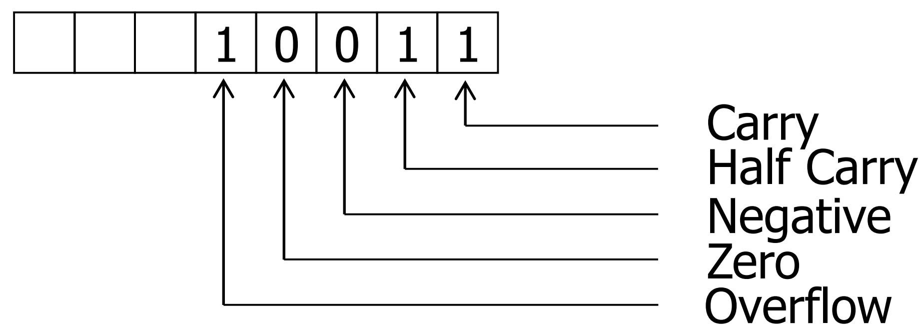
- C indicates a carry (or borrow) out of the most significant bit when an addition (or subtraction) is performed.
- V indicates two's complement (signed) overflow.
- Z indicates a result of exactly zero.
- N indicates a negative two's complement result (i.e., it's the MSB of the result).
- H indicates a carry out of bit 3. (Used only in implementing BCD arithmetic.)

Example

- A typical ADDA instruction and the resulting condition codes.

```
LDA A,$AA
ADD A,$BC
```

```
          1   <----------  H
   1 0 1 0 1 0 1 0
 + 1 0 1 1 1 1 0 0
 -----------------
 1 0 1 1 0 0 1 1 0
C  N
```



How CCR bits are set

- As a general rule, all CCR (status) bits are set by arithmetic instructions and by data transfer instructions.
- Branch instructions affect none of them. For precise information on all instructions, consult the instruction set table.
- The effect of each instruction on each CCR bit is described using the notation below.

Symbol	Operation
–	Bit unaffected
0	Bit always cleared (to 0)
1	Bit always set (to 1)
↕	Bit depends on result

Simple Branches

- There's a pair of simple branch instructions for each of the Z, V, C and H bits:
- BEQ, BNE branch on zero, branch on not zero
  - Branch if the Z bit is 1 or 0, respectively.
- BVS, BVC branch on V set, branch on V cleared
  - Branch if the V bit is 1 or 0, respectively. i.e., branch if two's complement overflow has or has not occurred.
- BCS, BCC branch on carry, branch on no carry
  - Branch if the C bit is 1 or 0, respectively.
- BHS, BHS branch on half carry, branch on no half carry
  - Branch if the H bit is 1 or 0, respectively.

Comparison Instructions

- These instructions perform a **subtraction** that sets the CCR bits according to the results and then throw the result away. They are used only for their effect on the CCR.
- CBA A, B compare B to A. Subtracts B from A
- C A, DATA
- C R_i, DATA
- C A, B
- C A, R_i
- C R_i, R_j
- CMP A, <ADDRESS>
- C R_{ii}, R_{jj}
- C R_{ii}, DATA DATA
- C R_{ii}, <ADDRESS>

Comparison Instructions

- Test: The TEST instruction is a special form of the **AND instruction** that produces no result except for a change of the flags.

TST A, V

TST R_i, V

TST A, B

TST A, R_i

TST R_i, R_j

TST A, <MEMORY>

TST R_i, <MEMORY>

Example

PS=$000A

PS					
0000	01 08 37 10 00	START	STA	$37,$1000	
0005	00 00 AA	REW	LDA	A, $AA	
0008	80 --		BR	REW	
000A	50 40		INC	A	
000C	80 --		BR	FOR	
000E	00 01 DD		LDA	B,$DD	
0011	C2		NOP		
0012	C2		NOP		
0013	40 10	FOR	MOV	C,A	
0015	1D 22 10 00		TST	C, <$1000>	

Example

```
PS
0000  01 08 37 10 00 | START  STA   $37,$1000
0005  00 00 AA       | REW    LDA   A, $AA
0008  80 FB          |        BR    REW
000A  50 40          |        INC   A
000C  80 --          |        BR    FOR
000E  00 01 DD       |        LDA   B,$DD
0011  C2             |        NOP
0012  C2             |        NOP
0013  40 10          | FOR    MOV   C,A
0015  1D 22 10 00    |        TST   C, <$1000>
```

PS=\$000A

Step = \$0005 - \$000A
= \$FB

Example

```
PS
0000   01 08 37 10 00   START   STA   $37,$1000
0005   00 00 AA         REW     LDA   A, $AA
0008   80 FB                    BR    REW
000A   50 40                    INC   A
000C   80 05                    BR    FOR
000E   00 01 DD                 LDA   B,$DD
0011   C2                       NOP
0012   C2                       NOP
0013   40 10            FOR     MOV   C,A
0015   1D 22 10 00              TST   C, <$1000>
```

PS=$000A

Step = $0005 - $000A
= $FB

PS = $000E

Step = $0013 - $000E
= $05

Comparison Branch Instructions

These fall into two distinct groups:

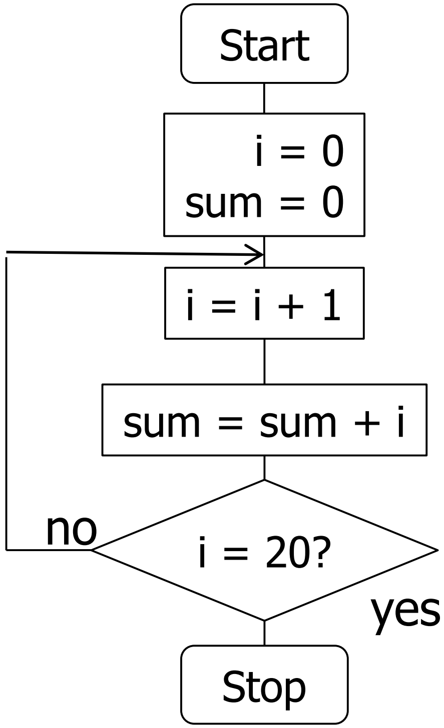
- The first is intended for use with signed (two's complement) data, and use the terminology greater, less, and equal.
  - BGT, BGE branch on greater than, greater than or equal to.
    - Branch if the result is > 00 or ≥ 00, respectively.
  - BLT, BLE branch on less than, less than or equal to.
    - Branch if the result is < 00 or ≤ 00, respectively.

- The second is intended for use with unsigned data, and use the terminology higher, lower, and same.
  - BHI, BHS branch on higher than, higher than or same.
    - Branch if the result is > 00 or ≥ 00, respectively.
  - BLO, BLS branch on lower than, lower than or same.
    - Branch if the result is < 00 or ≤ 00, respectively.

Comparison Branch Instructions

Example: Write a program to compute 1 + 2 + ... + 20 and save the sum at $1000.

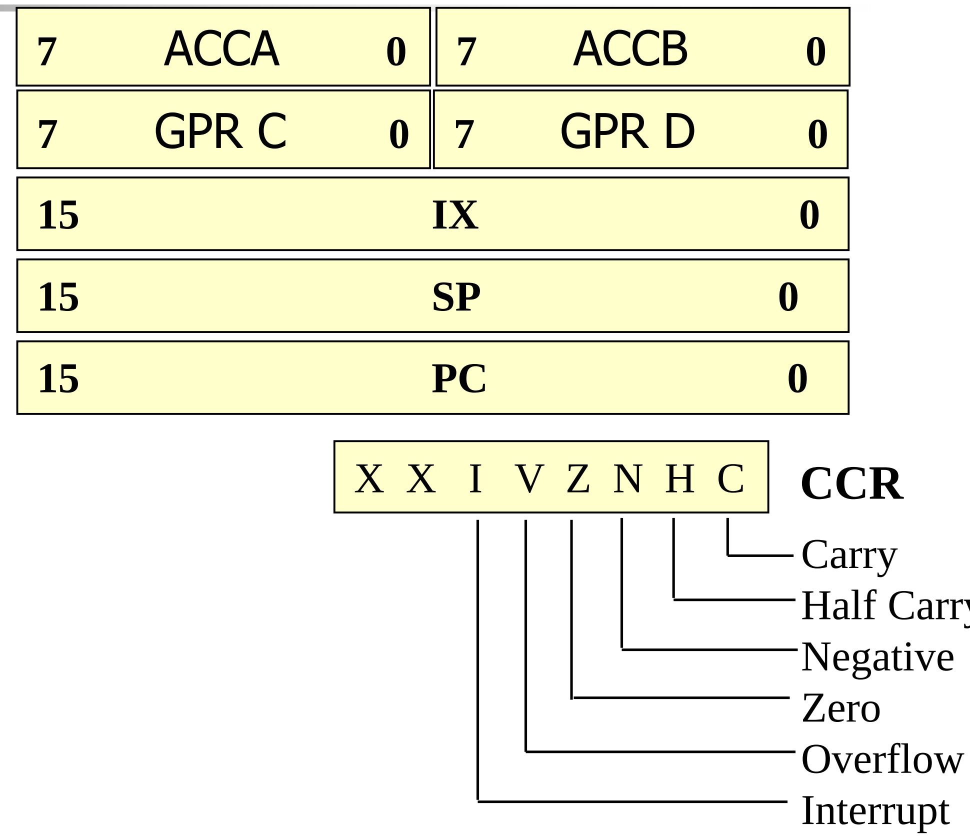
The following program uses accumulator B as the loop index i and ACCA as the sum.




```
START CLR   A
      CLR   B
AGAIN INC   B
      ADD   A, B
      CMP   B, $14
      BNEQ  AGAIN
      STA   A, $1000
      END
```

Features of the Educational CPU

- 8-bit Registers
  - ACCA
  - ACCB
  - GPR C
  - GPR D
  - CCR
- 16-bit Registers
  - AB
  - CD
  - IX
  - SP
  - PC



CCR Symbols:

0 Flag set to 0 as the result of the previous operation
1 Flag set to 0 as the result of the previous operation
-- Flag not affected as the result of the previous operation
↕ Flag can go either high or low as the result of the previous operation

Register Relative Addressing

- General purpose register pair CD contains the effective address
- Example:

```
START  LDA C,$10
       LDA D,$05
       STA $08,$1005
       LDA A,<CD>
```

At the end of the program: ACCA← $08

Addressing Codes

- Major Addressing Methods
  - Immediate Addressing (V)
  - Implied Addressing (Register Addressing) (L)
  - Direct Addressing (D)
  - Indirect Addressing (K)
  - Indexed Addressing (S)
  - Relative Addressing (B)
- Advanced Addressing Methods
  - Memory Immediate Write (V)
  - Incremented Index Addressing (R)
  - Decremented Index Addressing (Z)
  - Register Relative Index Addressing (U)
  - Stack Pointer Relative Addressing (Y)

Instruction Format of the EDU-CPU

a	b	c	d	e	f	g	h	k	n	o	p	s	u	v	y	Data/address	Data/address	address

1.Octal 2.Octal 3.Octal 4.Octal 5.Octal

a	b	Instruction Type
0	0	General
0	1	Inherent
1	0	Relative
1	1	Single octal

k	n	Operand for Inherent type
0	0	Two registers
0	1	Single register
1	0	CCR operation
1	1	Operation on a bit of register

c	word length
0	8 bit
1	16 bit

p	s	Operand for General type
0	0	Single register
0	1	Memory operation
1	0	Operation on a bit of memory
1	1	CCR related operation

k	n	o		Addressing Method
0	0	0	V	
0	0	1	D	
0	1	0	K	
0	1	1	S	
1	0	0	R	
1	0	1	Z	
1	1	0	U	
1	1	1	Y	

o	p	s		Register
u	v	y		
0	0	0	A	ACCA
0	0	1	B	ACCB
0	1	0	C	C
0	1	1	D	D
1	0	0	CCR	
1	0	1	IX	
1	1	0	SP	

o	p	s		Register Pair
u	v	y		
0	0	0	AB	
0	1	0	CD	

Example

- Seven numbers are stored in memory starting from address \$2000. Assume that the sum of these numbers can be stored in one memory location. Write a program to find the sum of the numbers and to store it at address \$5000. The program should begin at memory location \$1000.

Example

```
1000  4B 40              START  CLR   A
1002  00 01 07                  LDA   B, $07
1005  20 05 20 00               LDA   IX, $2000
1009  03 60 00           REW    ADD   A, <IX+00>
100C  70 45                     INC   IX
100E  51 41                     DEC   B
1010  82 F7                     BNEQ  REW
1012  01 20 50 00               STA   A, $5000
1016  C3                        SWI
```

*SWI : software interrupt