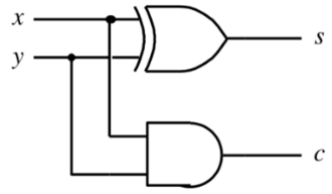
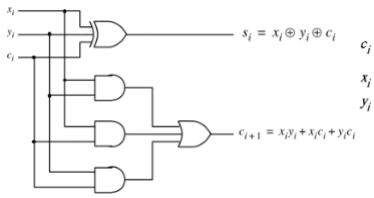
Half-Adder.

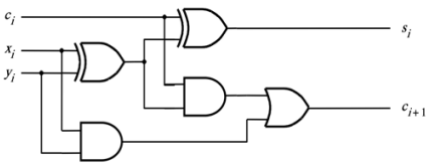


Representation:

Full-Adder



2-1MUX



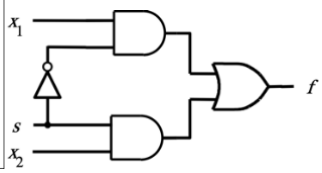
4-1MUX

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

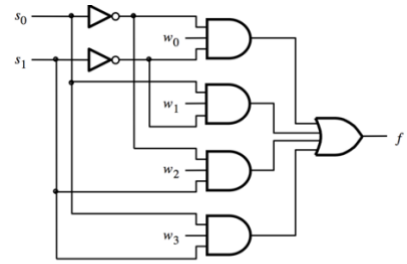
XOR(gives 1 when input diff)

$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	−0	−7	−8
1001	−1	−6	−7
1010	−2	−5	−6
1011	−3	−4	−5
1100	−4	−3	−4
1101	−5	−2	−3
1110	−6	−1	−2
1111	−7	−0	−1

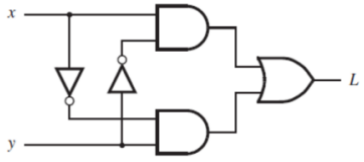
2-4 Decoder



3-8 Decoder



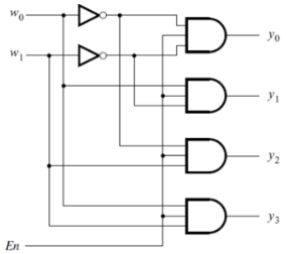
$f = \overline{s_1}\,\overline{s_0}\,w_0 + \overline{s_1}\,s_0\,w_1 + s_1\,\overline{s_0}\,w_2 + s_1\,s_0\,w_3$

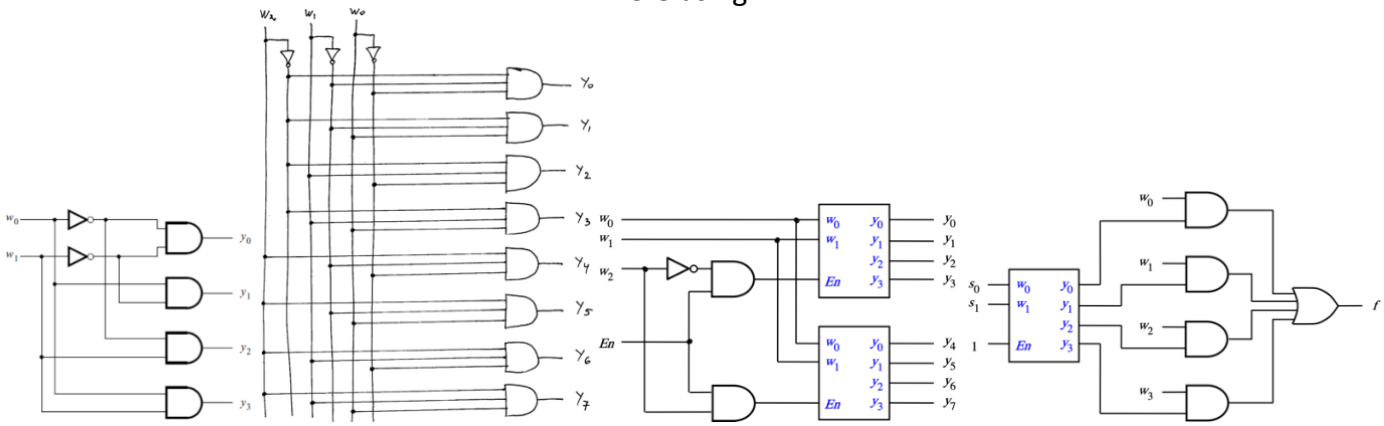


1-4 DeMUX w 2-4 decoder

w_1	w_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

3-8 using 2-4



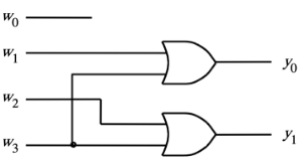


4-2 Binary Encoder

4-2 Priority

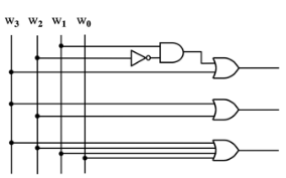
8-3 Binary encoder

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

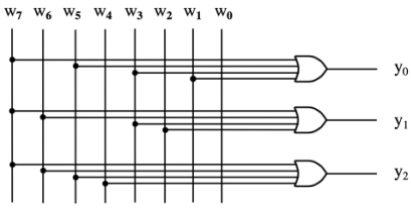


Shifter

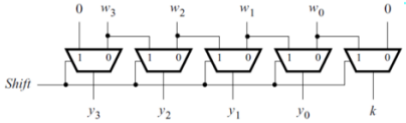
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1



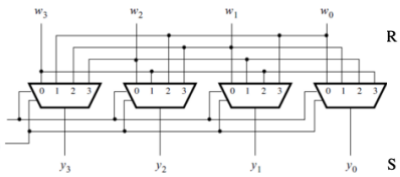
Barrel Shifter

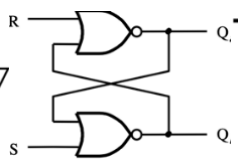


Basic Latch (w NOR)



s_1	s_0	y_3	y_2	y_1	y_0
0	0	w_3	w_2	w_1	w_0
0	1	w_0	w_3	w_2	w_1
1	0	w_1	w_0	w_3	w_2
1	1	w_2	w_1	w_0	w_3





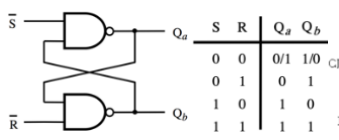
S	R	Q_a	Q_b
0	0	0/1	1/0
0	1	0	1
1	0	1	0
1	1	0	0

Basic Latch (w NAND).

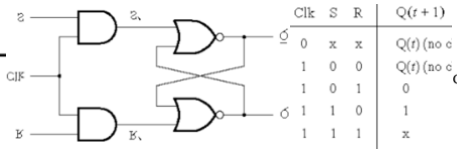
Gated SR Latch (w NOR)

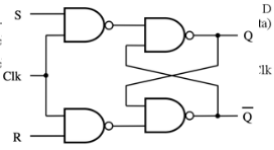
Gated SR Latch(w NAND)

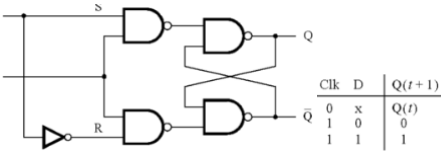
Gated D Latch- NAND



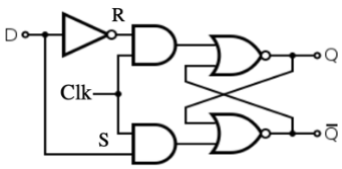
S	R	Q_a	Q_b
0	0	0/1	1/0
0	1	0	1
1	0	1	0
1	1	1	1





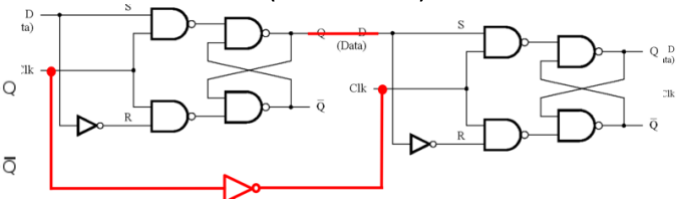


Gated D Latch- NOR.



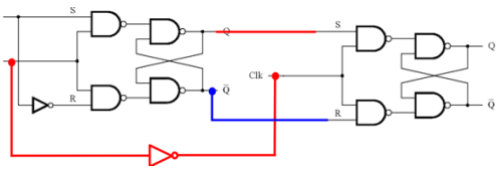
+ve DFF

Master slave DFF (2 D latches)

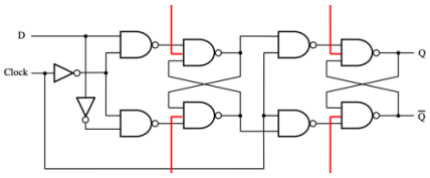


-ve DFF

Master slave DFF(1 D & 1 SR)

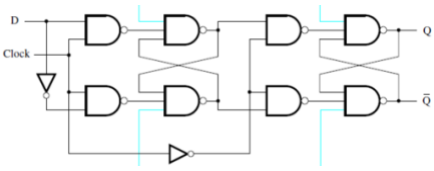


alternative DFF (+ve with only 6 NAND, fewer transistors)

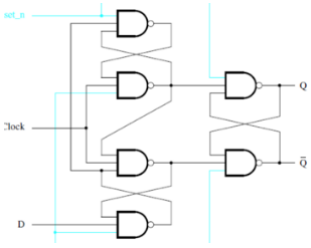


TFF

TFF(2-1 mux).

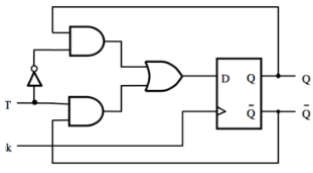


TFF (XOR)

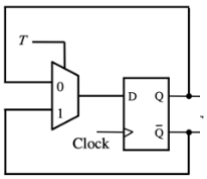


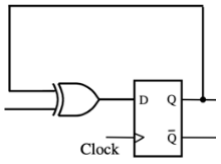
TT(TFF)

JK



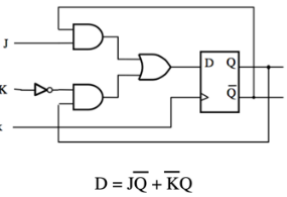
TFF(complete wire diag) +VE.





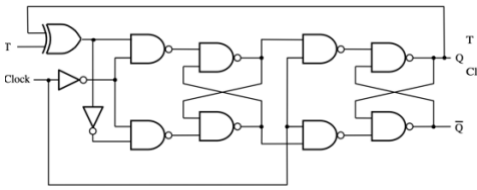
TFF(complete wire diag) -VE

T	$Q(t+1)$
0	$Q(t)$
1	$\overline{Q}(t)$

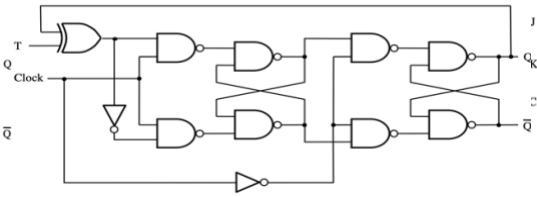


J	K	Q (t+1)	
0	0	Q (t)	Hold
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}$ (t)	Toggle

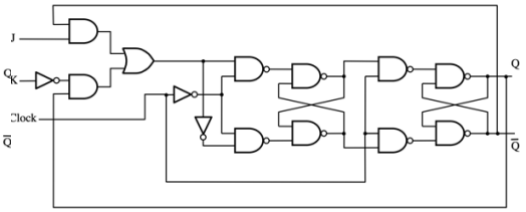
JK +VE complete



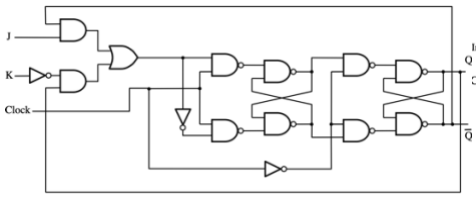
JK -VE complete



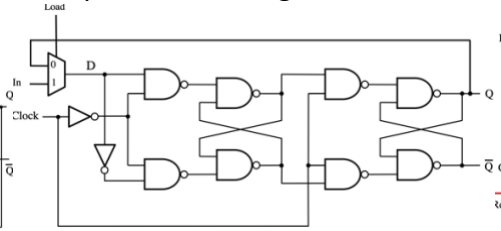
1-bit parallel acc. Register



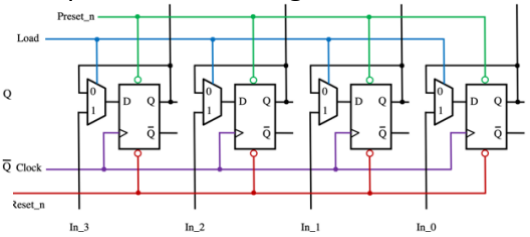
4-bit parallel access register



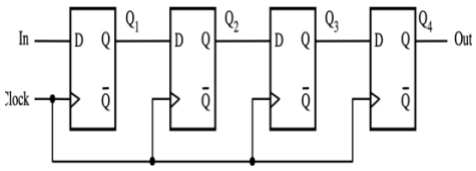
Shift Register



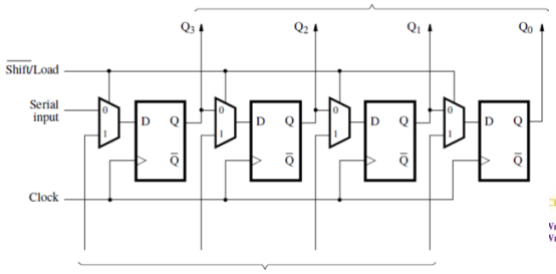
Parallel acc shift register.



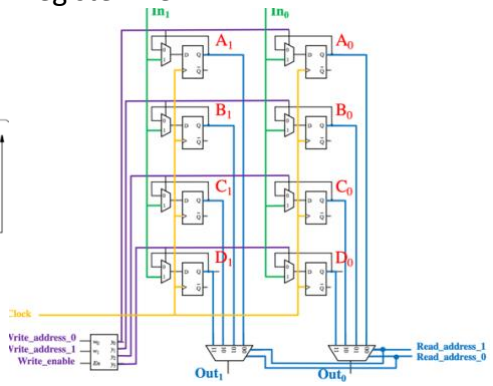
Register file



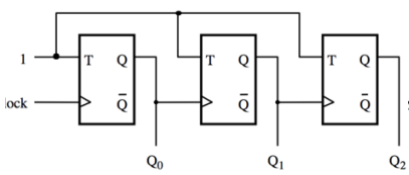
3-bit down counter.

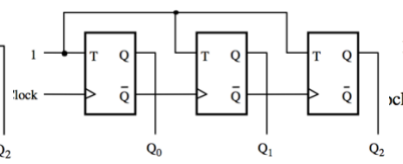


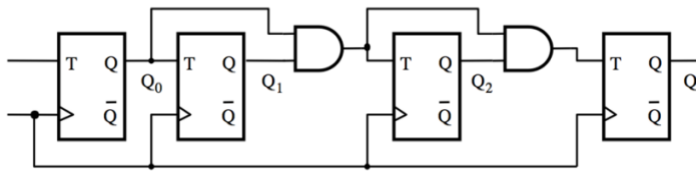
3-bit up counter



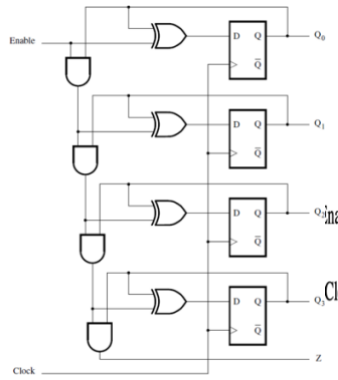
4-bit synch up counter



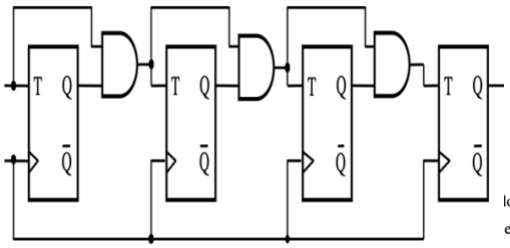




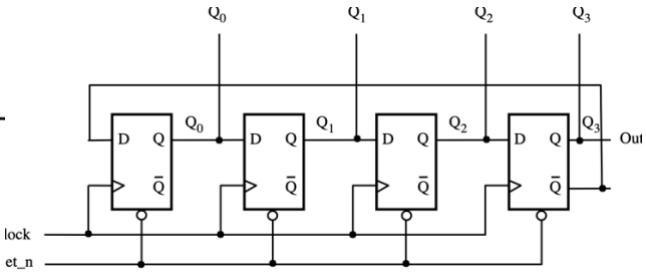
4-bit up counter(DFFs).



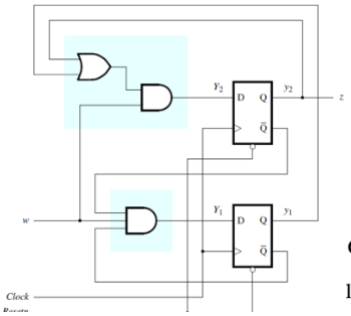
4-bit up counter(TFFs)



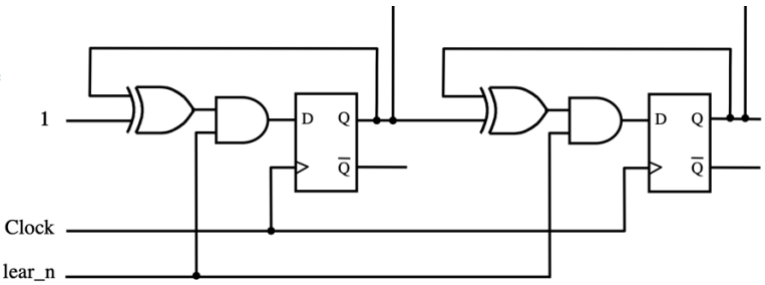
4-bit Johnson Counter



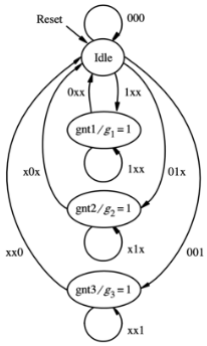
Example of diagram



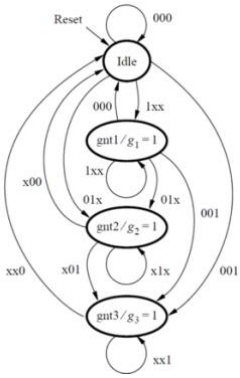
2-bit synch up counter(DFF)



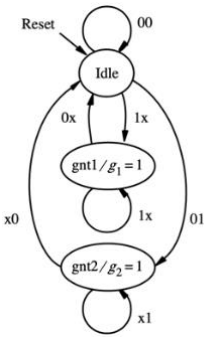
Arbiter Circuit



Arbiter Circuit (solved- 2 devices prob)



Simpler Arbiter.



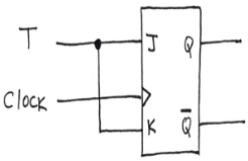
State Ass Table

Present state y_2y_1	Next state r_1r_2=00	01	10	11	Output g_1g_2
	Y_2Y_1				
A 00	00	10	01	01	00
B 01	00	00	01	01	10
C 10	00	10	00	10	01
11	dd	dd	dd	dd	dd

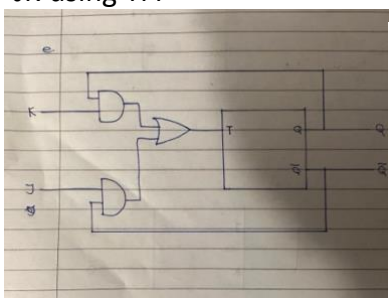
copying R1- R3 w R2 as temp

STEP	INSTRUCTION	REGISTER	GO TO STEP	[BRANCH TO STEP]
1.	Deb	3	1	2
2.	Deb	2	2	3
3.	Deb	1	4	6
4.	Inc	3	5	
5.	Inc	2	3	
6.	Deb	2	7	8
7.	Inc	1	6	
8.	End			

TFF using JK



JK using TFF



Assembly code for R=P+2Q

```
.data
P   BYTE 3      ;allocate space for 3 variables
Q   BYTE 2      ; and set them to their initial values
R   BYTE 5      ; P←3, Q←2, and R←5
.code
LOAD  A, [P]    ; load the value of P into register A
LOAD  B, [Q]    ; load the value of Q into register B
SHIFTL B        ; shift B left, effectively B←2*B
ADD   A, B      ; add regs. A and B, effectively A←P+2Q
STORE [R], A    ; store the result into R in the data memory
```

Assembly-User inputs 1 var

```
.data
x       BYTE 2
y       BYTE ?
result  BYTE ?

.code
        LOAD   A, [x]
        INPUTD [y]
        LOAD   B, [y]
        ADD    A, B
        STORE  [result], A
```

Assembly code(if x=y, x=z)

```
.data
P   BYTE 3      ;allocate space for 3 variables
Q   BYTE 2      ; and set them to their initial values
R   BYTE 5      ; P←3, Q←2, and R←5
.code
LOAD  A, [P]    ; load the value of P into register A
LOAD  B, [Q]    ; load the value of Q into register B
SHIFTL B        ; shift B left, effectively B←2*B
ADD   A, B      ; add regs. A and B, effectively A←P+2Q
STORE [R], A    ; store the result into R in the data memory
```

```
.data
x       BYTE 3
z       BYTE ?

.code
        LOAD   A, [x]
        MOVE   C, A      ; z=x;
        MOVE   B, A      ; B=x;
        SHIFTL B         ; B=2x
        SHIFTL B         ; B=4x
        ADD    C, B      ; C=4x+x
        STORE  [z], C    ; update
```

Assembly-Multiplication

```
.data
x       BYTE 3
z       BYTE ?

.code
        LOAD  A, [x]
        LOADI C, 0       ; z=0
        LOADI B, 0       ; i=0
        LOADI D, 5       ; sentinel value
For:    CMP   B, D       ; i<5?
        BRGE  End        ; if(i>=5), exit loo
        ADD   C, A       ; z+=x
        ADDI  B, 1       ; i++
        JUMP  For        ; jump to For loop
End:    STORE [z], C     ; update the z
                         ; value in memory
```

Loop for adding one var to another.

```
.data
N       BYTE 5
sum     BYTE ?

.code
        LOADI A, 0
        LOADI B, 0
        LOAD  D, [N]
Do:     ADDI  A, 1
        ADD   B, A
        CMP   D, A
        BRG   Do
End:    STORE [sum], B
```

Do loop for adding 1-5

Bubble sort

```
.data
array   BYTE 7, 3, 2, 1, 6, 4, 5, 8
last    BYTE 7
temp    BYTE ?

.code
        LOADI  A, 0              ; i = 0;
Outer:  LOAD   D, [last]         ; Load last into D
        LOADI  B, 0              ; j = 0;
        CMP    A, D              ; i < last
        BRGE   End               ; If i >= last bre
Inner:  LOAD   D, [last]         ; Re-Load last int
        SUB    D, A              ; D = D - A  (i.e.
        CMP    B, D              ; j < last - i
        BRGE   Iinc              ; If j >= last-i
If:     LOADF  C, [array+B]      ; C = array[j]
        LOADF  D, [array+B+1]    ; D = array[j+1] (
        CMP    D, C              ; if array[j+1] <
        BRGE   Jinc
Swap:   STOREF [array+B], D
        STOREF [array+B+1], C
Jinc:   ADDI   B, 1              ; j++
        JUMP   Inner
Iinc:   ADDI   A, 1              ; i++
        JUMP   Outer
End:    NOOP                     ; Do nothing
```

Array Plus 5

```
.data
array   BYTE    1, 2, 3, 4
N       BYTE    4
.code
        LOADI   A, 0            ; i = 0
For:    LOAD    D, [N]          ; D <- N
        CMP     A, D            ; i < N ?
        BRGE    End             ; if no, exit l
        LOADF   C, [array + A]  ; load array[i]
        ADDI    C, 5            ; add 5
        STOREF  [array + A], C  ; store the res
Iinc:   ADDI    A, 1            ; i++
        JUMP    For             ; next iteratic
End:    NOOP
```

IF w 2 conditions(AND)

```
.data
x       BYTE    5
min     BYTE    1
max     BYTE    8
inRange BYTE    0

.code
        LOAD  A, [x]
        LOAD  B, [min]
        LOAD  C, [max]
        CMP   B, A          ; min <= x
        BRG   End
        CMP   A, C          ; x <= max
        BRG   End
        LOADI D, 1
        STORE [inRange], D
End:    NOOP
```

IF-Else(x<y,z=x else z=y)

```
.data
x       BYTE    3
y       BYTE    5
z       BYTE    ?

.code
        LOAD  A, [x]
        LOAD  B, [y]
If:     CMP   A, B
        BRGE  Else
        STORE [z], A
        JUMP  End
Else:   STORE [z], B
End:    NOOP
```

IF Greater

```
.data
x       BYTE    3
y       BYTE    5
z       BYTE    0

.code
        LOAD  A, [x]
        LOAD  B, [y]
        CMP   B, A
        BRGE  End
        STORE [z], A
End:    NOOP
```

If greater or equal

```
.data
x       BYTE    3
y       BYTE    5
z       BYTE    0

.code
        LOAD  A, [x]
        LOAD  B, [y]
        CMP   B, A
        BRG   End
        STORE [z], A
End:    NOOP
```

IF not equal

```
.data
x       BYTE    3
y       BYTE    5
z       BYTE    0

.code
        LOAD  A, [x]
        LOAD  B, [y]
        CMP   A, B
        BRE   End
        STORE [z], A
End:    NOOP
```

IF Equal

```
.data
x       BYTE    3
y       BYTE    5
z       BYTE    0

.code
        LOAD  A, [x]
        LOAD  B, [y]
        CMP   A, B
        BRNE  End
        STORE [z], A
End:    NOOP
```

If less or equal

```
.data
x       BYTE    3
y       BYTE    5
z       BYTE    0

.code
        LOAD  A, [x]
        LOAD  B, [y]
        CMP   A, B
        BRG   End
        STORE [z], A
End:    NOOP
```

IF less

```
.data
x       BYTE    3
y       BYTE    5
z       BYTE    0

.code
        LOAD  A, [x]
        LOAD  B, [y]
        CMP   A, B
        BRGE  End
        STORE [z], A
End:    NOOP
```

Unsigned v Signed.

• Equal	ZF	
• Not equal	$\overline{ZF}$	
• Greater	$\overline{ZF}$ • XNOR(NF, OF)	
• Greater or Equal	XNOR(NF, OF)	
• Less	XOR(NF, OF)	
• Less or Equal	ZF + XOR(NF, OF)	

• Equal	ZF
• Not equal	$\overline{ZF}$
• Above	$\overline{ZF}$ • $\overline{CF}$
• Above or Equal	$\overline{CF}$
• Below	CF
• Below or Equal	ZF + CF

I281