1.) DATA TRANSFER

- MOVE X, Y (SET CON(Y) TO CON(X))
- LOAD X (COPY CON(X) INTO REGISTER R)
- STORE X (COPY CONTENTS OF REGISTER R INTO MEMORY CELL X.)

2.) ARITHMETIC

- ADD_1 X (SET CON(R) TO CON(X) + CON(R))
- ADD_2 X, Y (SET CON(Y) TO CON(X) + CON(Y))
- ADD_3 X, Y, Z (SET CON(Z) TO CON(X) + CON(Y))

3.) COMPARE

THE RESULT OF A COMPARE OPERATION IS TO SET THE VALUES OF SPECIAL BITS CALLED CONDITION CODES: GT, EQ, LT

- COMPARE X, Y (COMPARE CON(X) TO CON(Y) AND SET CONDITION CODES ACCORDINGLY)

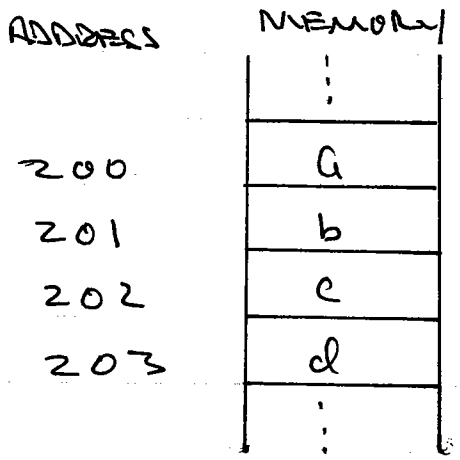
IF	GT	EQ	LT
CON(X) > CON(Y)	1	0	0
CON(X) = CON(Y)	0	1	0
CON(X) < CON(Y)	0	0	1

4.) BRANCHING

THESE INSTRUCTIONS ALTER THE NORMAL SEQUENTIAL FLOW OF PROGRAM EXECUTION.

- JUMP X (TAKE NEXT INSTRUCTION FROM CELL X.)
- JUMPGT X (IF GT = 1 TAKE NEXT INSTRUCTION FROM CELL X, OTHERWISE FOLLOW SEQUENCE.)
- JUMPEQ X
- JUMPLT X
- JUMPGE X
- JUMPLE X
- JUMPNEQ X
- HALT

IN THE FOLLOWING EXAMPLES, ASSUME THAT THE VALUES a, b, c, d ARE STORED IN CELLS 200, 201, 202, 203 RESPECTIVELY.



Ex.

WRITE MACHINE LANGUAGE INSTRUCTIONS TO:
SET a TO b + c.

ADDRESS	MEMORY	COMMENTS
	⋮	
100	LOAD 201	PUT CON(201) INTO R
101	ADD$_1$ 202	PUT CON(202)+CON(R) INTO R
102	STORE 200	PUT CON(R) INTO 200
	⋮	

OR

	⋮	
100	ADD$_2$ 201,202	PUT CON(201)+CON(202) IN 202
101	MOVE 202,200	PUT CON(202) IN 200
	⋮	(NOTE: c is lost)

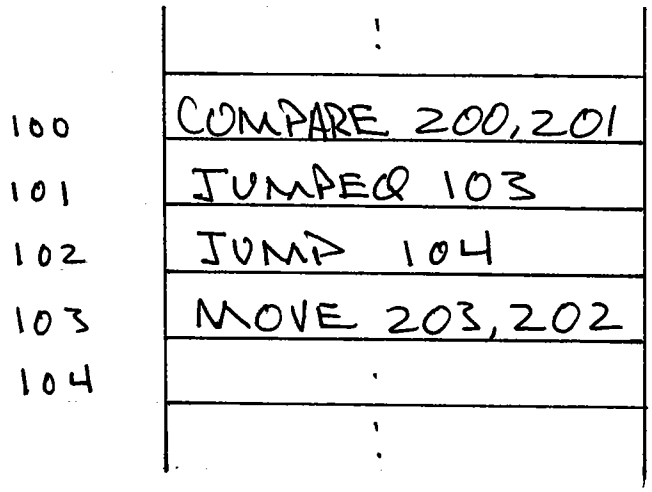
OR

	⋮	
100	ADD$_3$ 201,202,200	PUT CON(201)+CON(202) INTO 200
	⋮	(NOTE: c NOT lost.)

THIS ILLUSTRATES THE POINT THAT A COMPLEX INSTRUCTION SET LEADS TO SHORTER PROGRAMS, AND CONVERSELY A SIMPLER INSTRUCTION SET LEADS TO LONGER PROGRAMS.

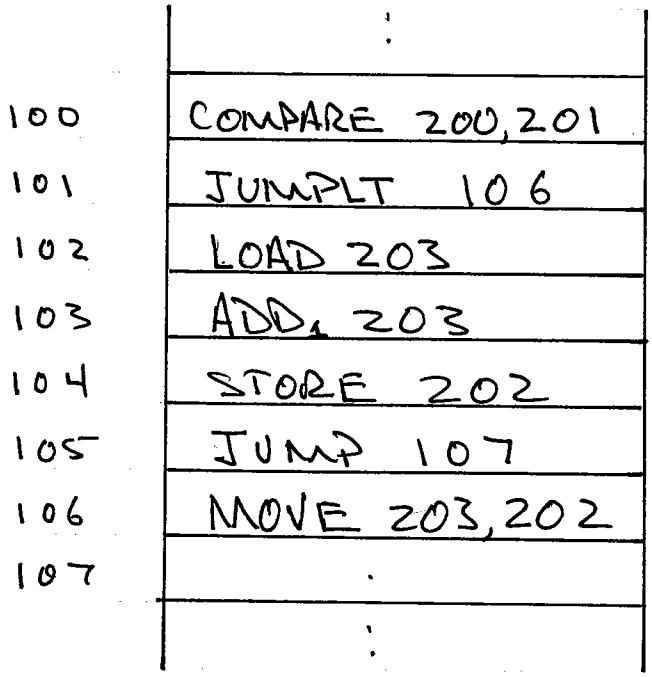
EX.

IF a = b SET c TO d



EX.

IF a < b SET c TO d
ELSE SET c TO 2d

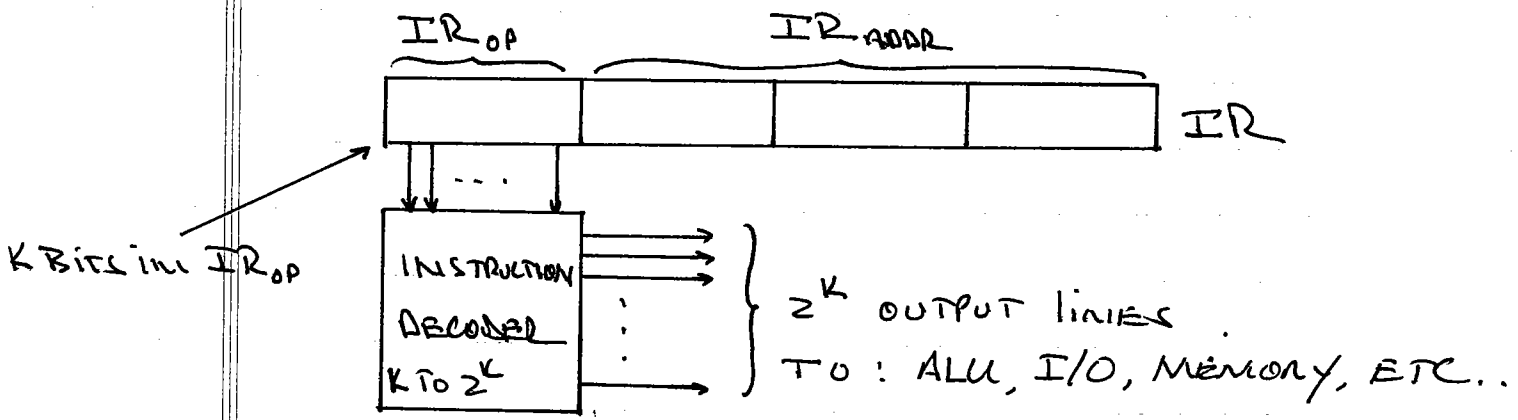


EX.

REPEAT UNTIL a > c
    SET a TO a + b
END LOOP.

	:		
100	LOAD	201	PUT CON(201) IN R
101	ADD	200	PUT CON(200)+CON(R) IN R
102	STORE	200	PUT CON(R) IN 200
103	COMPARE	200,202	
104	JUMPGT	106	
105	JUMP	100	
106			
	:		

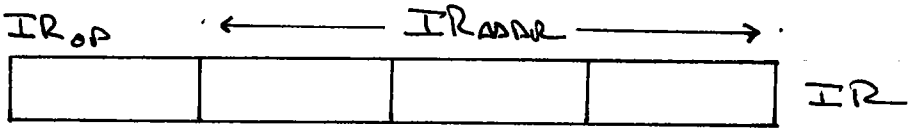
IN ADDITION TO THE INSTRUCTION REGISTER THE CONTROL UNIT CONTAINS THE INSTRUCTION DECODER AND THE PROGRAM COUNTER.



THE INSTRUCTION DECODER TAKES THE K BITS IN IR_{OP} AND INTERPRETS THEM AS A BINARY NUMBER IN THE RANGE 0 TO $2^K - 1$. A 1 IS PLACED ON THE CORRESPONDING OUTPUT LINE, WHICH ACTIVATES THE CIRCUITRY NEEDED TO PERFORM THE DESIRED OPERATION.

THE PROGRAM COUNTER HOLDS THE ADDRESS OF THE NEXT INSTRUCTION TO BE EXECUTED. IT IS INCREMENTED AFTER EACH INSTRUCTION STEP BY THE NUMBER OF BYTES IN A SINGLE INSTRUCTION.

EX.



#BITS: 8 24 24 24

EACH INSTRUCTION OCCUPIES 80 BITS = 10 BYTES IN MEMORY.

FIG. 5.18 ON P. 203 SHOWS THE OVERALL ORGANIZATION OF THE SUBSYSTEMS IN THE VON NEUMANN ARCHITECTURE.

ALTHOUGH OUR DISCUSSION OF THE VON NEUMANN ARCHITECTURE HAS BEEN SOMEWHAT SUPERFICIAL, WE HAVE COME QUITE FAR.

WE HAVE SEEN HOW THE MOST ELEMENTARY DEVICES (TRANSISTORS) CAN BE USED TO BUILD COMPLEX SUBSYSTEMS (GATES, CONTROL CIRCUITS, ADDERS, ETC..) WHICH CAN THEN BE USED TO IMPLEMENT OUR ALGORITHM STEPS.

FIG. 5.19 P. 204 SHOWS THE INSTRUCTION SET FOR A VERY SIMPLE VON NEUMANN COMPUTER. WE WILL USE THIS INSTRUCTION SET IN THE NEXT PROGRAMMING ASSIGNMENT.

IN PA3 YOU WILL WRITE SIMULATED MACHINE LANGUAGE PROGRAMS FOR A SIMULATED PROCESSOR.

- READ: 5.3 HISTORICAL OVERVIEW P. 208-228

HW 6: CHAP 5 P. 228:
3, 4, 5, 8, 11, 17, 18 abcd