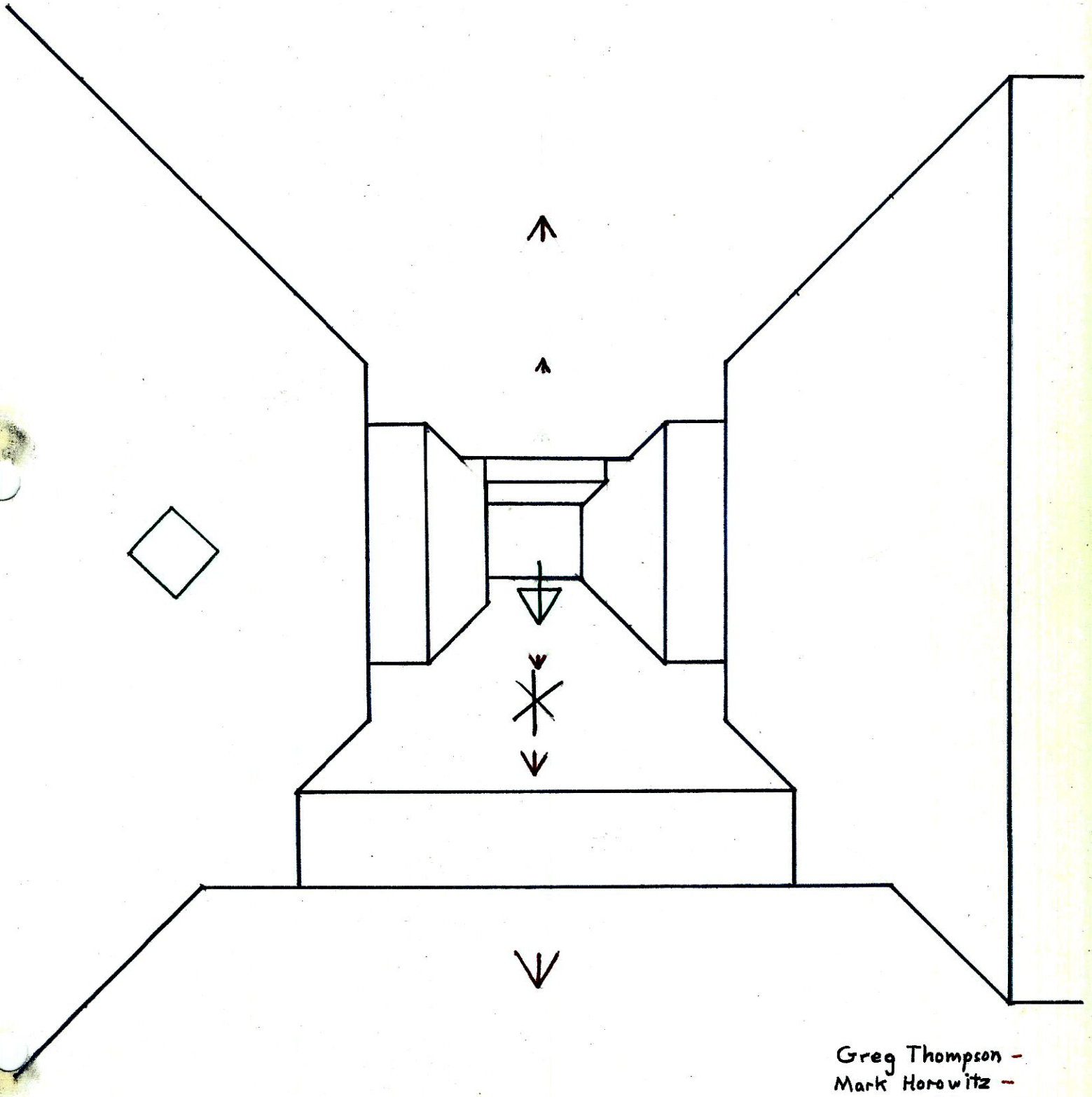
Design 9.9
Report 9
Demo 10

The MAZE game



Greg Thompson -
Mark Horowitz -
George Waltman -

6.111 , 6.112
Fall 1976

The MAZE game

person	main responsibility
Greg Thompson	Maze processor hardware
George Woltman	Maze processor software
Mark Horowitz	Maze display processor

6.111, 6.112
Digital Project Lab, M.I.T.

Fall 1976

Table of Contents

Display Processor

Maze Overview

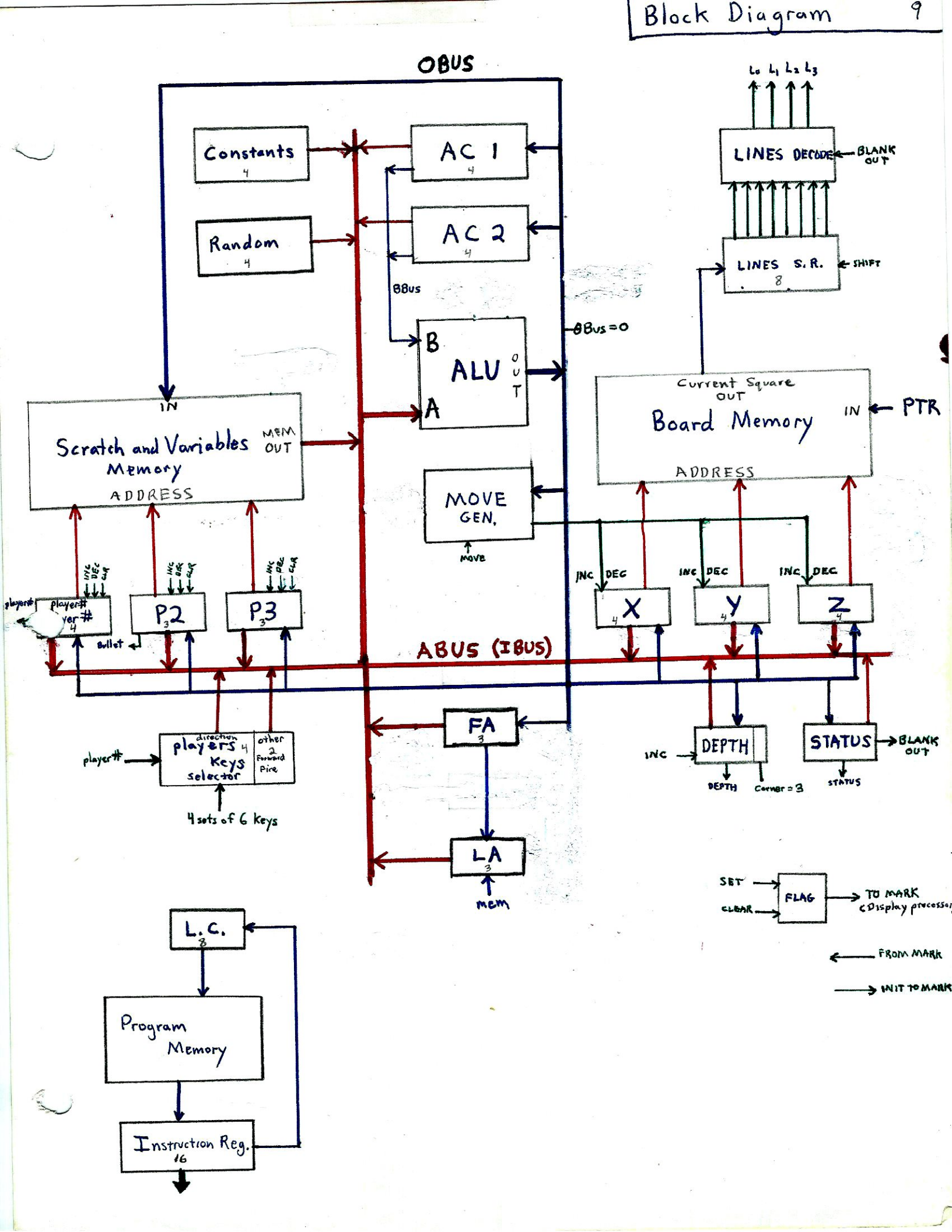
A maze is implemented by a 16x16x16 cube, where some of the subcubes are filled in. The open subcubes form passageways through the large cube and are the corridors of the maze. In this maze are placed X (1→4) people and 8-X (7→4) robots. Each person has a set of switches, which he uses to move through the maze, and an oscilascope which displays his current view in the maze. In addition to switches that effect the players position (forward, turn right, left, up, down) there is a fire switch that launches a bullet down the corridor that the player is currently facing. The object of the game is to walk through the maze and shoot the other players before they shoot you. To make the game a little more interesting robots were added. They act the same as the other people in the maze except they are machince controlled. Their speed is adjustable which allows you to vary their ability.

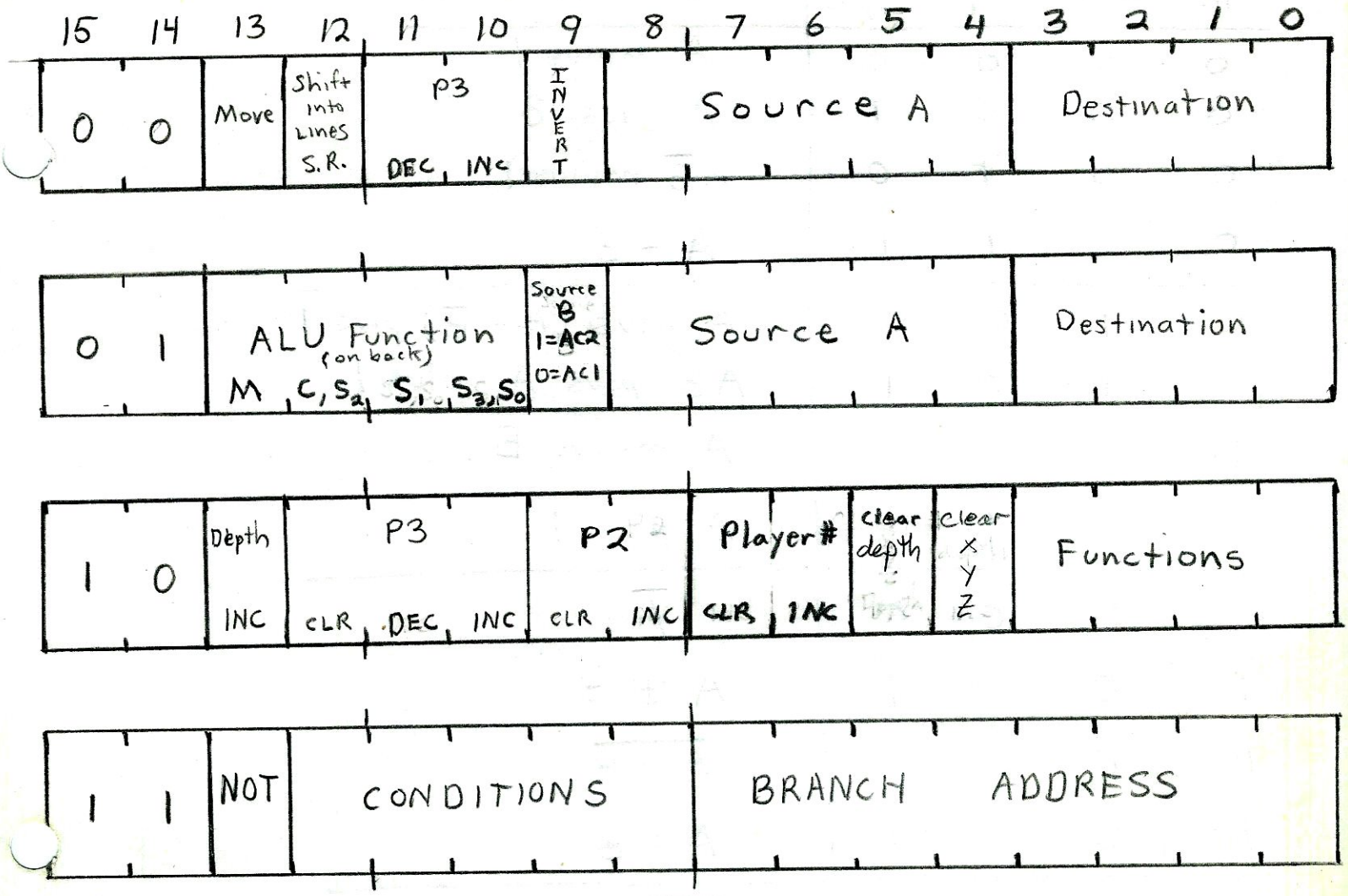
MAZE PROCESSOR

The Maze game hardware is organized as a general four bit processor optimized to execute the maze game software. There are 256 - 16 bit words in the program memory implemented in 1702 proms. There are two RAM memories used in the processor; the board memory, where the format of the current maze is kept and the Scratch and Variables memory where status, location and orientation information is kept for the players, bullets and robots. Both memories are three dimensionally addressed; the board memory by the X, Y and Z registers (which represent a location in the maze) and the scratch memory by the player#, P2 and P3 registers. The board memory can be loaded from a paper tape reader using a standard ASCII encoded tape. Four bits along the X axis are loaded per character from the low order four bits of the tape. Thus @ through ? can be used load four empty through four filled squares respectively. If the 2nd high order bit is on or the 4th high order bit is off the character is ignored allowing NULS, CRs, LFs, spaces and RUBOUTS to be used for formatting and tape editting. The output of the board memory is a single bit (current_square) which is high (1) if the current square in the maze (addressed by the X, Y and Z registers) is filled in and low (0) if not. The current square line can be tested by a conditional branch instruction and is used as input to a 8-bit lines shift-register and lines decoding logic. The lines decoding logic determines if each of the four lines emminating from a given corner can be seen or not which is determined combinatorially from the status of the seven squares surrounding the corner (the square the player is in or viewing through is assumed to empty). A blank screen line, which is high if any of the three low order bits of the status register are one, causes all four lines to be low and handles the cases where you been shot and haven't re-entered the maze yet, are not an active player, or can't see the lines because there is a filled square closer to you in the cooridor blocking your view.

The depth register is used to keep track of where we are during generation of the views down the cooridors. Axises are encoded using the three low order bits of the fourbit word as follows: X-100, Y-010, Z-001, -X-011, -Y-101, -Z-110. This encoding has the advantages that the negative of an axis is just the ones complement of an axis, it only takes one three input XOR to determine if an axis is positive or negative and it only takes one two input XOR to determine if an axis is along any given X, Y or Z axis. By placing an axis valve in the FA (forward axis) register and having a second axis (down axis) available at the outputs of the scratch memory the LA (left axis) logic will determine the third orthagonal axis. With an axis valve on the output bus of the ALU (OBUS) the move generator logic will move forward one square along that axis by incrementing or decrementing the X, Y, or Z registers appropriately. There are two general purpose accumulators (AC1 and AC2) available that input to either the A or B side of the ALU over the ABUS or BBUS respectively. The ABUS is also known as the input bus or IBUS since almost all the registers and the output of the scratch memory may input to that BUS. The output of the ALU, the OBUS, can be stored in any of 12 places including memory and the two general registers AC1 and AC2. Available on the IBus are all the possible constants (0 to 15), a random value (used when placing a player in the maze and by the robot routine), 4 sets of 6 players keys, the output of memory, the address registers: Player#, P2, P3, X, Y and Z, FA, LA the depth and status registers. There are four basic instructions in the maze processor. The first is the source → destination or $\overline{\text{source}}$ → destination instruction with bits to control the move generator, shifting current-square into the lines shift register and INCing or DECing the P3 address register. The second instruction is the ALU function instruction: ALU(ABUS, BBUS) → destination. Four bits are used to encode the six inputs to the ALU (M, C, S_0, S_1, S_2 and S_3) giving us a 16 instruction subset of those possible on the 74181 chip. The third instruction is a combination of misc. functions and clearing and

incrementing functions on the address and depth registers of the processor. The final instruction is the conditional branch with 32 possible conditions, a bit to invert (NOT) the condition and a 8-bit branch address in the right half of the instruction. The display and control panel displays the current instruction about to be executed, the value of the ready line to the display processor, current_square, the value of the location counter (which always points to the next instruction), an address stop light which is on if the current instruction location matches that in the address stop switch register, a $\overline{\text{Branch}}$ light which is off if a branch instruction is about to branch, the values of the AC1, AC2, player#, P2, and P3 registers, the current output of memory (addressed by player#, P2 and P3) and the value on the OBUS and ABUS (NOTED in the decimal point lights). Switches are provided for RUN/single cycle mode, Enable/Disable address stop and to input the # of robots in the game (from 4 to 7 robots). There are three push-buttons: RESET (which after two clock pulses will reset the location counter to zero), a single cycle button (warning: pressing the single cycle button disables address stop) and a branch overide switch which manually NOTs the conditions of branch instructions. Because the processor is simultaneously fetching the next instruction as it is executing the current one an extra cycle is need if the current instruction is a branch instruction and it wishes to branch.


OBUS
Constants
4
Random
4
AC 1
4
AC 2
4
BBus
B
ALU
A
OUT
BBus=0
IN
Scratch and Variables Memory
ADDRESS
MEM OUT
MOVE GEN.
MOVE
Current Square OUT
Board Memory
ADDRESS
IN
PTR
L0 L1 L2 L3
LINES DECODE
BLANK OUT
LINES S.R.
8
SHIFT
INC DEC CLR
player#
P2
P3
bullet
INC DEC
X
Y
Z
ABUS (IBUS)
FA
3
LA
3
MEM
player#
direction players Keys selector
4
other
2
Forward Fire
4 sets of 6 keys
INC
DEPTH
Corner=3
STATUS
BLANK OUT
SET
CLEAR
FLAG
TO MARK (Display processor)
FROM MARK
INIT TO MARK
L.C.
8
Program Memory
Instruction Reg.
16



Source A

0		16 - Random
1		17 - AC1
2	C	18 - AC2
3		19 - memory
4	O	20 - X
5	N	21 - Y
6		22 - Z
7	S	23 - player#
8	T	24 - P2
9	A	25 - P3
10		26 - FA
11	N	27 - LA
12	T	28 - Depth
13		29 - Status
14	S	30 - Direction Keys
15		31 - Other Keys

Destination

0 -	none
1 -	AC1
2 -	AC2
3 -	memory
4 -	X
5 -	Y
6 -	Z
7 -	player #
8 -	P2
9 -	P3
10 -	FA
11 -	
12 -	Depth
13 -	Status
14 -	
15 -	

Conditions

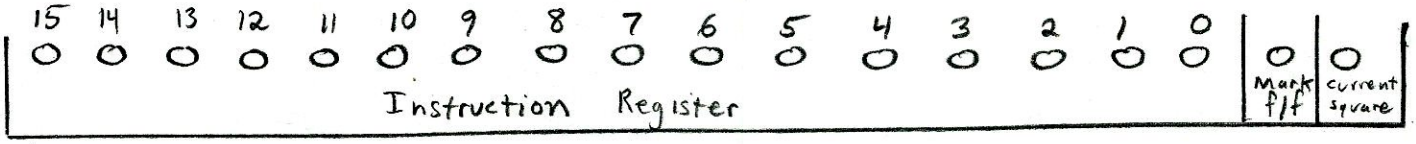
0	never	16	FA7 = ± X
1	char_recieved	17	FA7 = ± Y
2	Tape channel 402	18	FA7 = ± Z
3	OBvs = 0	19	AC2 ≥ Depth
4	P3 bit 3	20	Bullets Ready
5	P2 bit 3	21	Refresh Ready
6	player3 bit 3	22	Robots Ready
7	player bit 2	23	SWin < player#
8	line from Mark	24	AC1$_0$
9	depth_corner7 = 3	25	AC1$_1$
10	Status bit 3	26	AC1$_2$
11	Random bit 1	27	AC1$_3$
12	status (1+2)	28	mem = AC1
13	Depth bit 7	29	mem bit 1
14	Mem = 0	30	AC2 bit 3
15	FA positive	31	current square

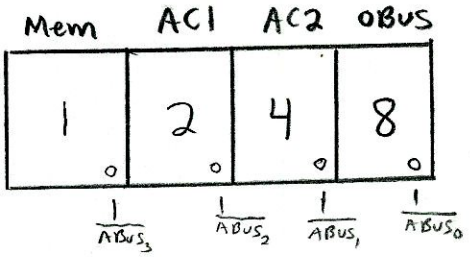
Functions

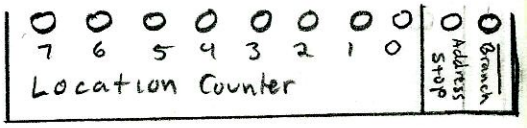
0	(no function)
1	
2	set Mark flag
3	get PTR
4	write_board_mem
5	refresh reset and init Mark
6	- bullets reset
7	- robot reset
8	
9	
10	
11	
12	
13	
14	
15	

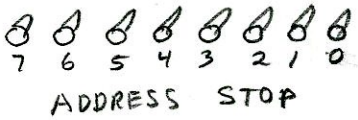
M	C, S_2	S_1	S_3, S_0	ALU Function
0	0	0	0	A minus 1
0	0	0	1	A plus B
0	0	1	0	$A\overline{B}$ minus 1
0	0	1	1	A + B
0	1	0	0	A plus $(A + \overline{B})$ plus 1
0	1	0	1	AB plus A plus 1
0	1	1	0	A minus B
0	1	1	1	A plus 1
1	0	0	0	$\overline{A}$
1	0	0	1	$A \oplus B$
1	0	1	0	$\overline{A + B}$
1	0	1	1	A + B
1	1	0	0	$\overline{A + B}$
1	1	0	1	$A\overline{B}$
1	1	1	0	$\overline{A \oplus B}$
1	1	1	1	A

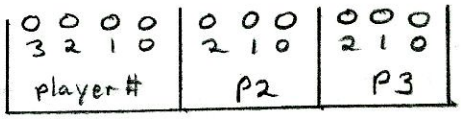
Control and Display console

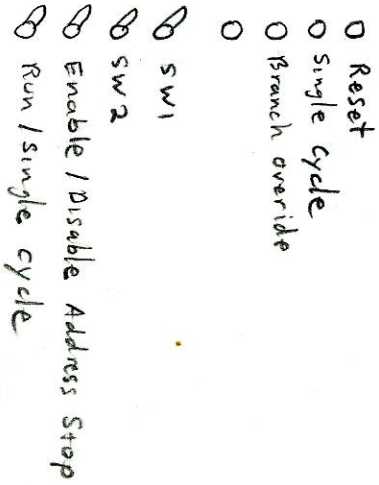






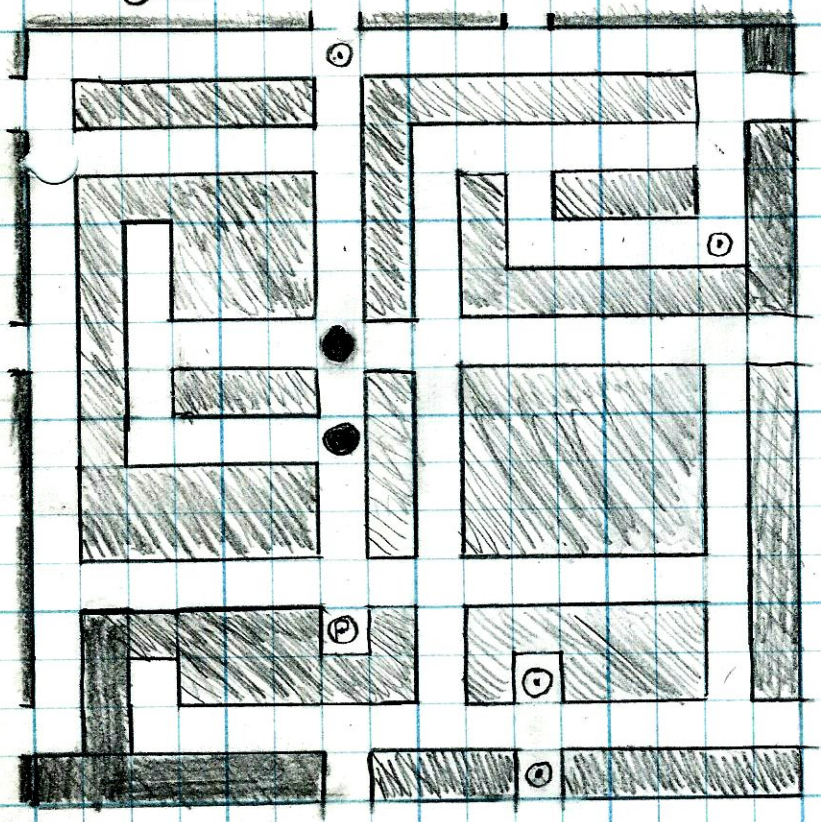


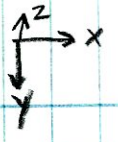




Bus Drivers 643	MOVE ORS 532	MOVE Selector MSI 154	Lines NORS 502	V
P3 MSI 193	Bus Drivers 643	Z MSI 193	Lines NANDS 500	U
P2 MSI 193	Bus Drivers 643	Y MSI 193	Lines NANDS 500	T
Player # MSI 193	Bus Drivers 643	X MSI 193	Lines XORS MSI 86	S
Scratch + Variable Memory LSI 2102	J/K flip flops 570	Board Memory LSI 2102	LINES S.R. MSI 95	R
Bus Drivers 643	XORs MSI 86	memory pull-ups 600A	Square selector MSI 153	Q
mem and AC1 comparator MSI 85	4-input NANDS 520	Bus Driver 643	UART board LSI 1602	P
Swin and Player # Comparator MSI 85	4-input NORS 525	FA register MSI95	Bet-PTR / Flag MSI 122	O
Address Stop Comparator MSI 85	9 latches 650	XORS MSI 86	AC1 Register MSI 95	N
Switches	Reset, clock, Address Stop Logic 500	AC2 Register MSI 95	Bus Driver 643	M
Numbers	Clock / Flag MSI 122	AC2 < Depth Comparator MSI 85	Bus Driver 643	L
Led Lamps	Clock / Flag MSI 122	Depth Register MSI 193	Bus Driver 643	K
B selector MSI 157	clock / oBus Driver 540	Decode ORS 532	Status Reg MSI 95	J
Binput inverters 645	A-Bus pull ups + Logic 1s 600A	Decode NANDS 500	Bus Driver 643	H
ALU MSI 181	Condition selector #4 MSI 151	Decode NANDS 500	Random Reg MSI 163	H
Func Selector MSI 157	condition selector #3 MSI 151	Decode NANDS 500	Bus Drivers 643	G
18 I.R. LAMPS 691	condition selector #2 MSI 151	conditional NORS 502	Keys Selector MSI 157	F
LEFT-HALF I.R. MSI 95	Condition Selector #1 MSI 151	Source Decode NORS 502	Keys Selector MSI 157	E
LEFT-HALF CODE MEMORY LSI 1702	I.R. Decode Buffer Drivers 540	Source Decode NORS 502	Keys selector MSI 157	D
LOCATION COUNTER MSI 163	IR Decode Marks flag NOR 502	Source Decode NORS 502	players Keys pull-ups 600A	C
RIGHT-HALF CODE MEMORY LSI 1702	Source Decoder MSI 154	Destination Decode NORS 502	players Keys pull-ups 600	B
RIGHT-HALF I.R. MSI 95	Destination Decoder MSI 154	Destination Decode NORS 502	Teletype board	A
21	k	m	4	

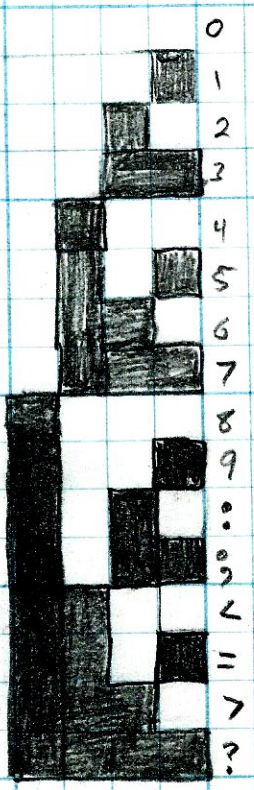
1st floor Demonstration Maze



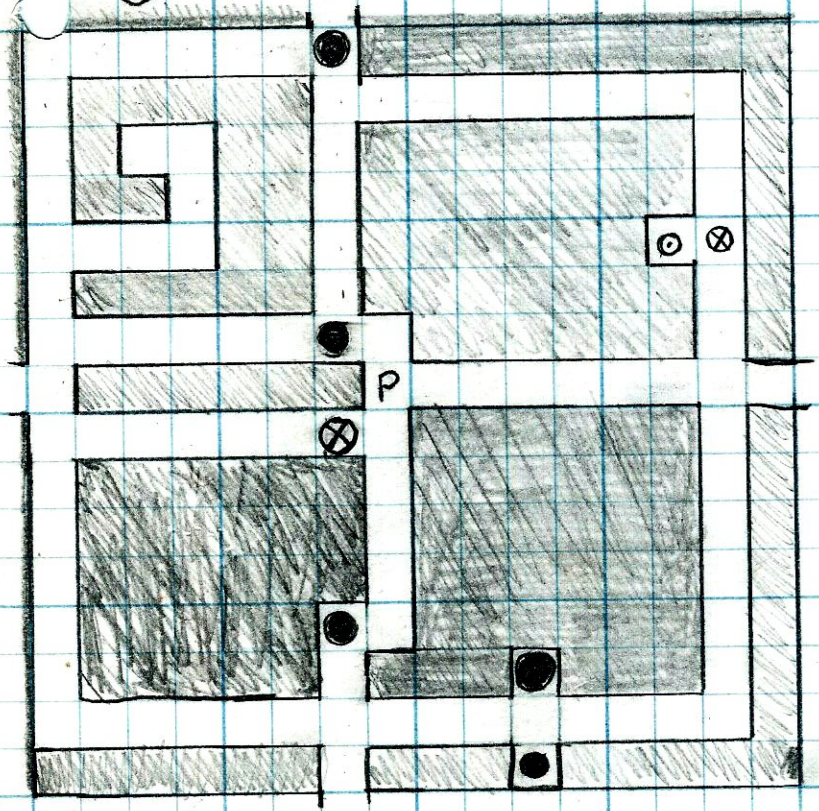


z=0

0	0	0	1
7	=	?	<
0	1	0	1
7	=	5	=
5	=	4	1
5	=	7	?
4	0	0	0
5	=	7	=
4	1	7	=
7	=	7	=
7	=	7	=
0	0	0	1
7	=	7	=
5	?	5	=
4	0	0	0
?	=	=	?



2nd floor



z=1,2

?	=	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	=
?	?	?	?
?	=	?	?
?	?	?	?
?	=	?	?
?	?	?	?
?	?	?	?
?	=	?	?
?	?	=	?
?	?	?	?
?	?	=	?

> on z=2

z=3

0	1	?	?
7	<	0	1
4	=	?	=
6	=	?	=
0	=	?	9
7	=	?	=
0	0	?	=
7	>	0	0
0	0	?	=
7	>	?	=
7	>	?	=
7	>	?	=
7	<	?	=
7	=	=	=
0	0	0	1
?	=	=	?

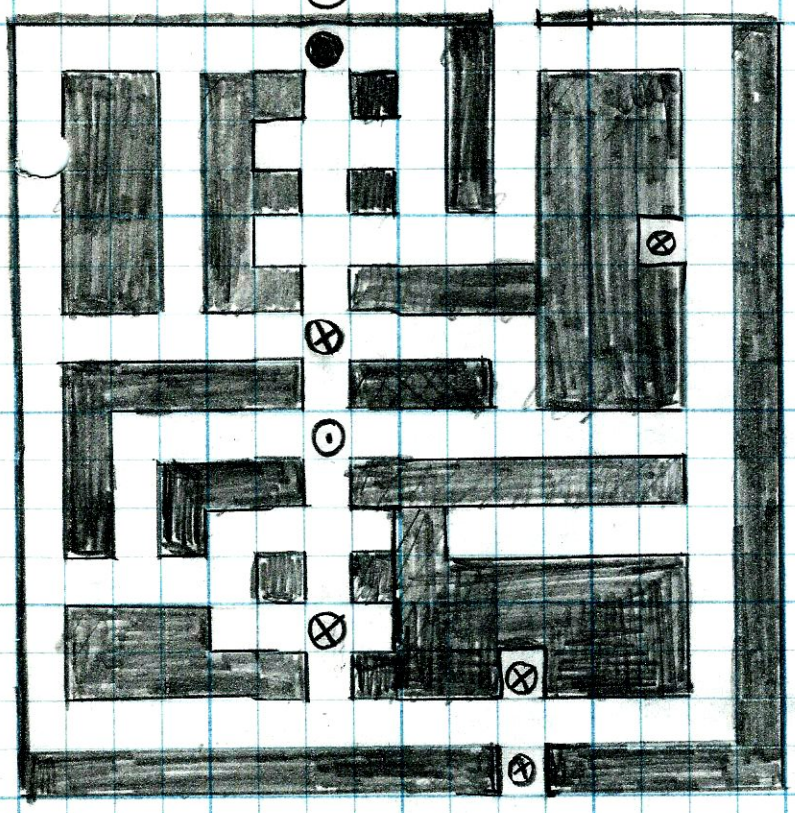
⊙ - up shaft

⊗ - down shaft

● - up and down shaft

P - pit

③rd floor



z=4,5,6

?	=	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	;
?	?	?	?
?	=	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	=	?	?
?	?	=	?
?	?	?	?
?	?	=	?

z=7

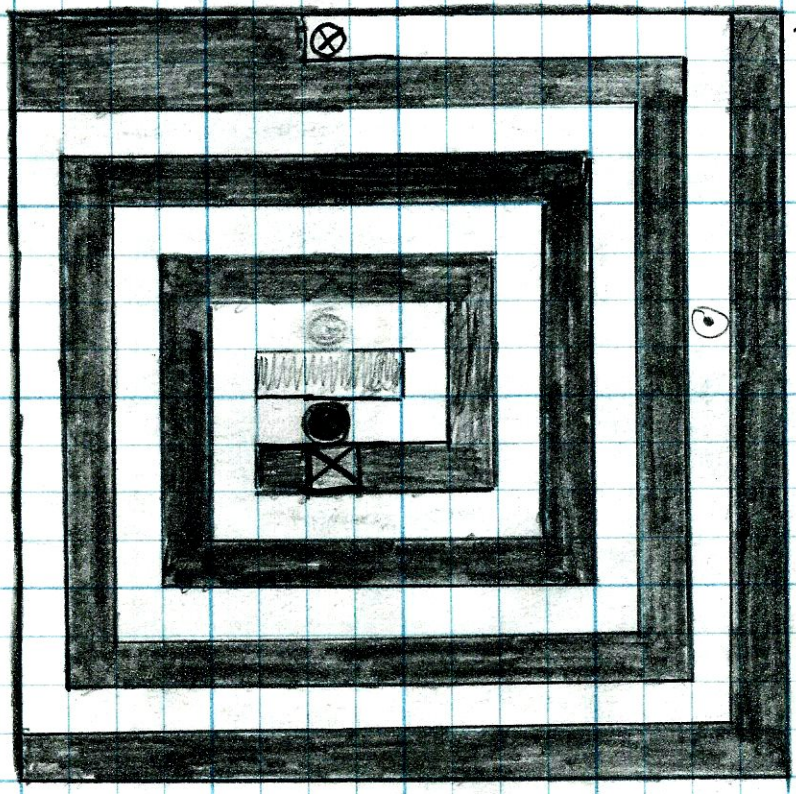
0	0	4	1
6	=	5	=
6	8	5	=
6	=	5	=
6	8	1	9
6	=	?	=
0	0	1	=
7	=	=	=
4	0	0	1
5	=	?	=
5	0	8	1
0	5	?	=
7	0	?	=
7	=	=	=
0	0	0	1
?	?	=	?

z=8,9

?	=	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	=	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

= on z=9

④th floor



z=10

?	<	0	1
?	?	?	=
0	0	0	5
7	?	?	5
4	0	1	5
5	?	=	5
5	0	5	5
5	7	5	5
5	0	5	5
5	7	=	5
5	0	1	5
5	?	?	5
4	0	0	5
7	?	?	5
0	0	0	1
?	?	?	?

z=11, 12, 13, 14, 15

?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	=	?	?
?	?	?	?
?	=	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?
?	?	?	?

= on z=12, 13, 14, 15

Board list

6 - 500 (NANDs)
8 - 502 (NORs)
1 - 520 (4-input NANDs)
1 - 525 (4-input NORs)
2 - 532 (ORs)
2 - 540 (NAND buffers)
1 - 570 (dual J/K f/fs)
~~3~~ 1 - 600A (pull-ups)
11 - 643 (bus-drivers)
1 - 645 (INANDs)
1 - 650 (9-latches)
4 - MSI 85 (comparator)
3 - MSI 86 (XORs)
7 - MSI 95 (shift-reg)
3 - MSI 122 (monostables)
4 - MSI 151 (8 to 1 selectors)
1 - MSI 153 (4 to 1 selectors)
3 - MSI 154 (4 to 16 decoders)
5 - MSI 157 (2 to 1 selectors)
2 - MSI 163 (registers)
1 - MSI 181 (ALU)
7 - MSI 193 (up/down registers)
2 - LSI 2102 (RAM)
2 - LSI 1702 (PROM)
1 - LSI 1602 (UART)

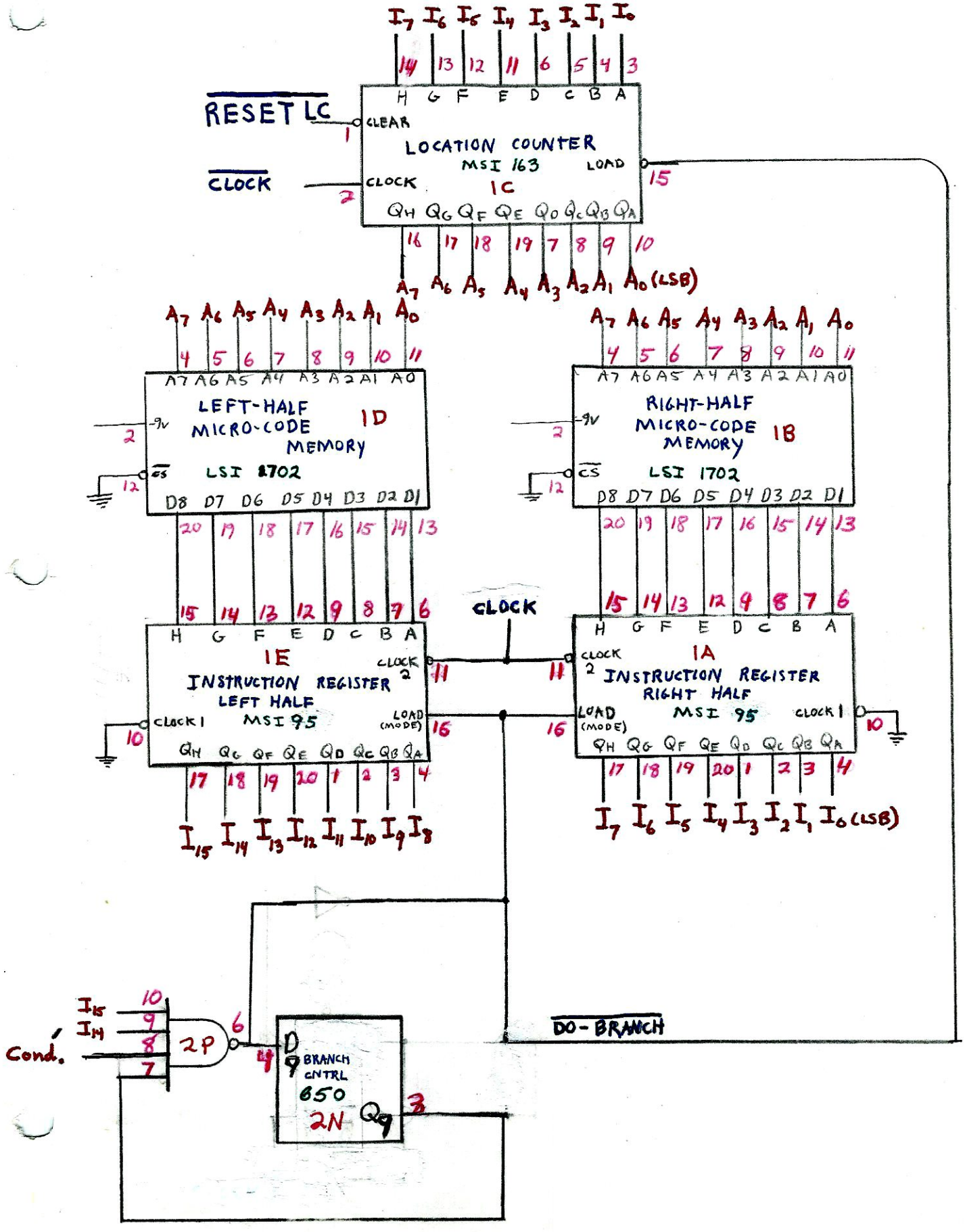
83

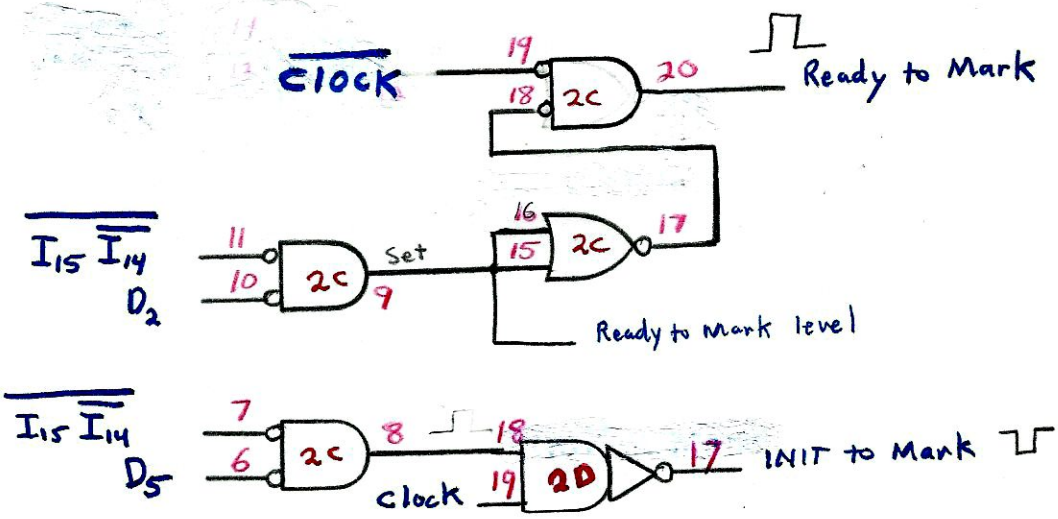
Spare logic

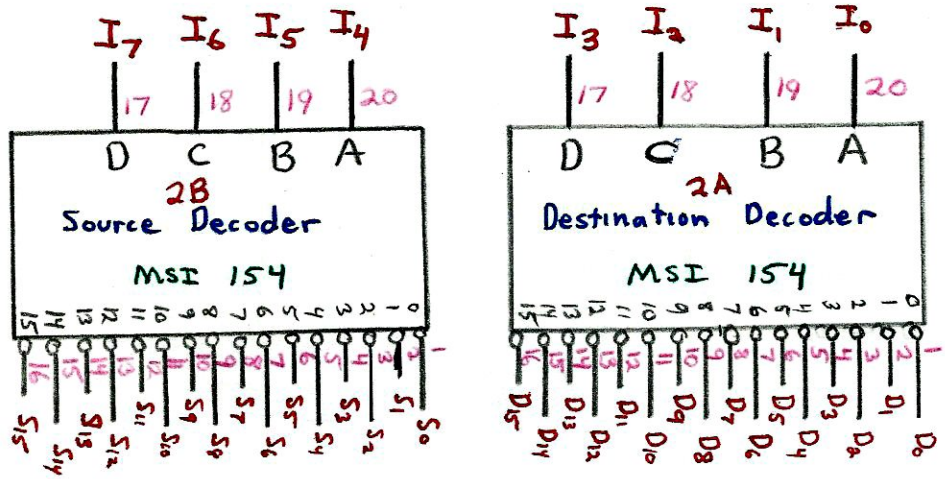
4V - 1 NOR
4U - 2-NAND
4T - 2 NAND
~~8A - 1 inverter~~
1I - 2 NAND
2P - 2- 4 input NANDs
2T - 4- 643 Bus drivers
2V, 3J - buffers (1 input ORs)

Hardware

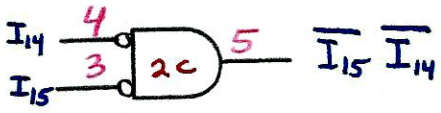
6 rails (22 slots per rail)
Power supplies to drive 6 rails (+5V)
-9V supply for PROMs
+/- 15 supply for display gen. DACs
paper tape reader (teletype)

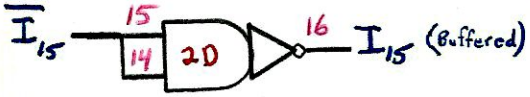

I7 I6 I5 I4 I3 I2 I1 I0
14 13 12 11 6 5 4 3
H G F E D C B A
RESET LC
1
CLEAR
LOCATION COUNTER
MSI 163
LOAD
CLOCK
CLOCK
2
1C
15
QH QG QF QE QD QC QB QA
16 17 18 19 7 8 9 10
A7 A6 A5 A4 A3 A2 A1 A0 (LSB)
A7 A6 A5 A4 A3 A2 A1 A0
4 5 6 7 8 9 10 11
A7 A6 A5 A4 A3 A2 A1 A0
LEFT-HALF
MICRO-CODE
MEMORY
1D
-9V
2
CS
12
LSI 1702
D8 D7 D6 D5 D4 D3 D2 D1
20 19 18 17 16 15 14 13
RIGHT-HALF
MICRO-CODE
MEMORY
1B
-9V
2
CS
12
LSI 1702
D8 D7 D6 D5 D4 D3 D2 D1
20 19 18 17 16 15 14 13
15 14 13 12 9 8 7 6
H G F E D C B A
1E
INSTRUCTION REGISTER
LEFT HALF
CLOCK 2
11
CLOCK 1
MSI 95
LOAD (MODE)
16
10
QH QG QF QE QD QC QB QA
17 18 19 20 1 2 3 4
I15 I14 I13 I12 I11 I10 I9 I8
CLOCK
15 14 13 12 9 8 7 6
H G F E D C B A
1A
CLOCK 2
11
INSTRUCTION REGISTER
RIGHT HALF
LOAD (MODE)
16
MSI 95
CLOCK 1
10
QH QG QF QE QD QC QB QA
17 18 19 20 1 2 3 4
I7 I6 I5 I4 I3 I2 I1 I0 (LSB)
I15
I14
Cond.
10
9
8
7
2P
6
D
4
9
BRANCH
CNTRL
650
2N
Q9
3
DO-BRANCH

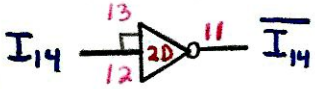

CLOCK
19
18
2C
20
Ready to Mark
$\overline{I_{15}}\ \overline{I_{14}}$
D_2
11
10
2C
9
Set
16
15
2C
17
Ready to Mark level
$\overline{I_{15}}\ \overline{I_{14}}$
D_5
7
6
2C
8
18
clock
19
2D
17
INIT to Mark

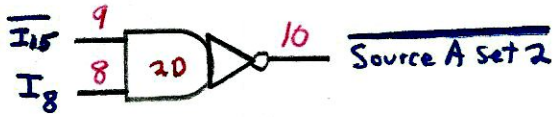

I_7 I_6 I_5 I_4
17 18 19 20
D C B A
2B
Source Decoder
MSI 154
I_3 I_2 I_1 I_0
17 18 19 20
D C B A
2A
Destination Decoder
MSI 154

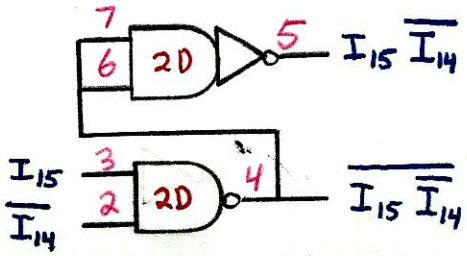

I_{15}
2C
$\overline{I_{15}}$


I_{14}
I_{15}
4
3
2C
5
$\overline{I_{15}}\ \overline{I_{14}}$

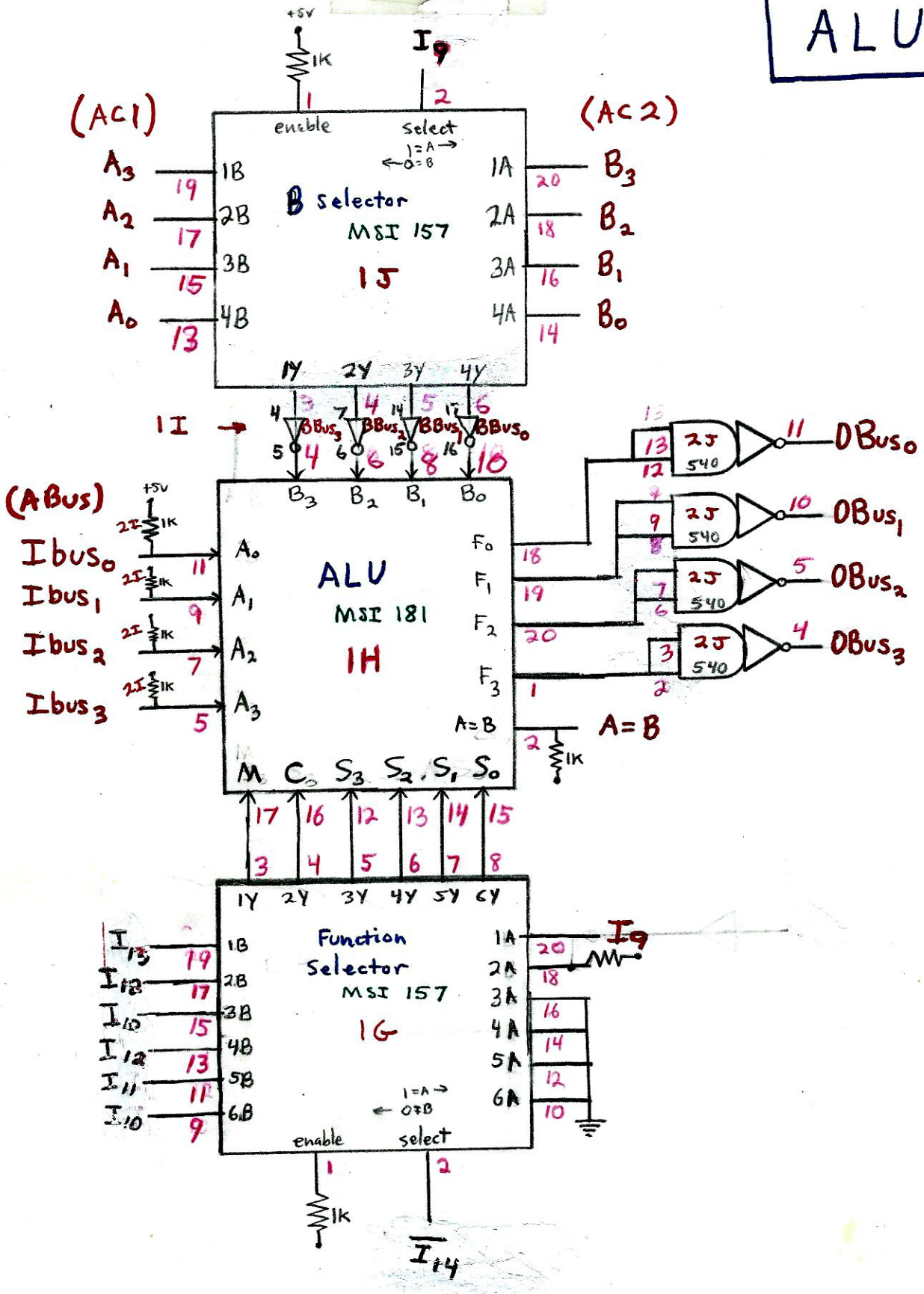

$\overline{I_{15}}$
15
14
2D
16
I_{15} (Buffered)

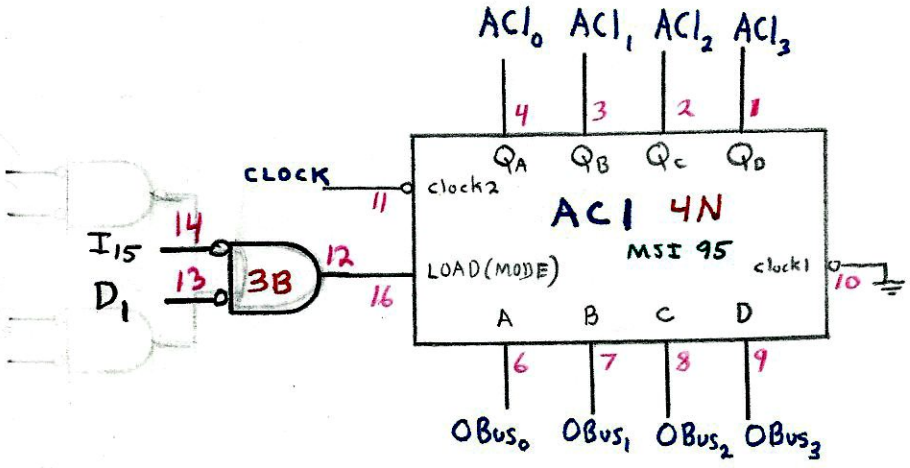

I_{14}
13
12
2D
11
$\overline{I_{14}}$

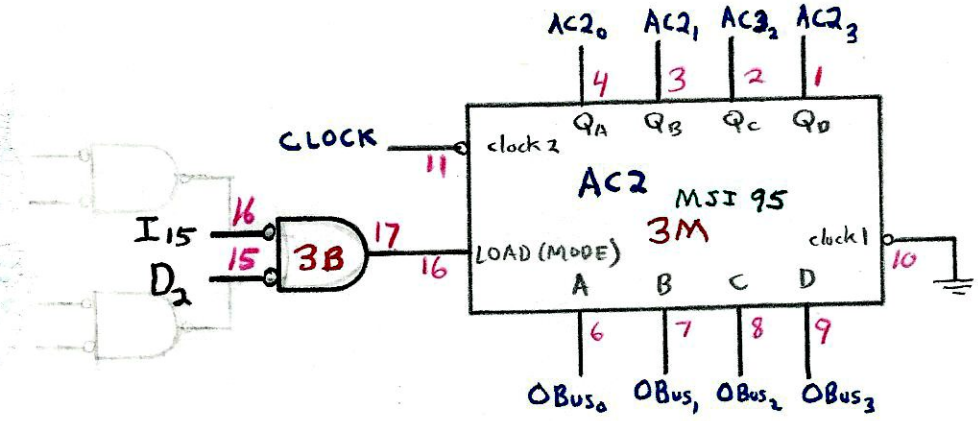

$\overline{I_{15}}$
I_8
9
8
2D
10
Source A Set 2

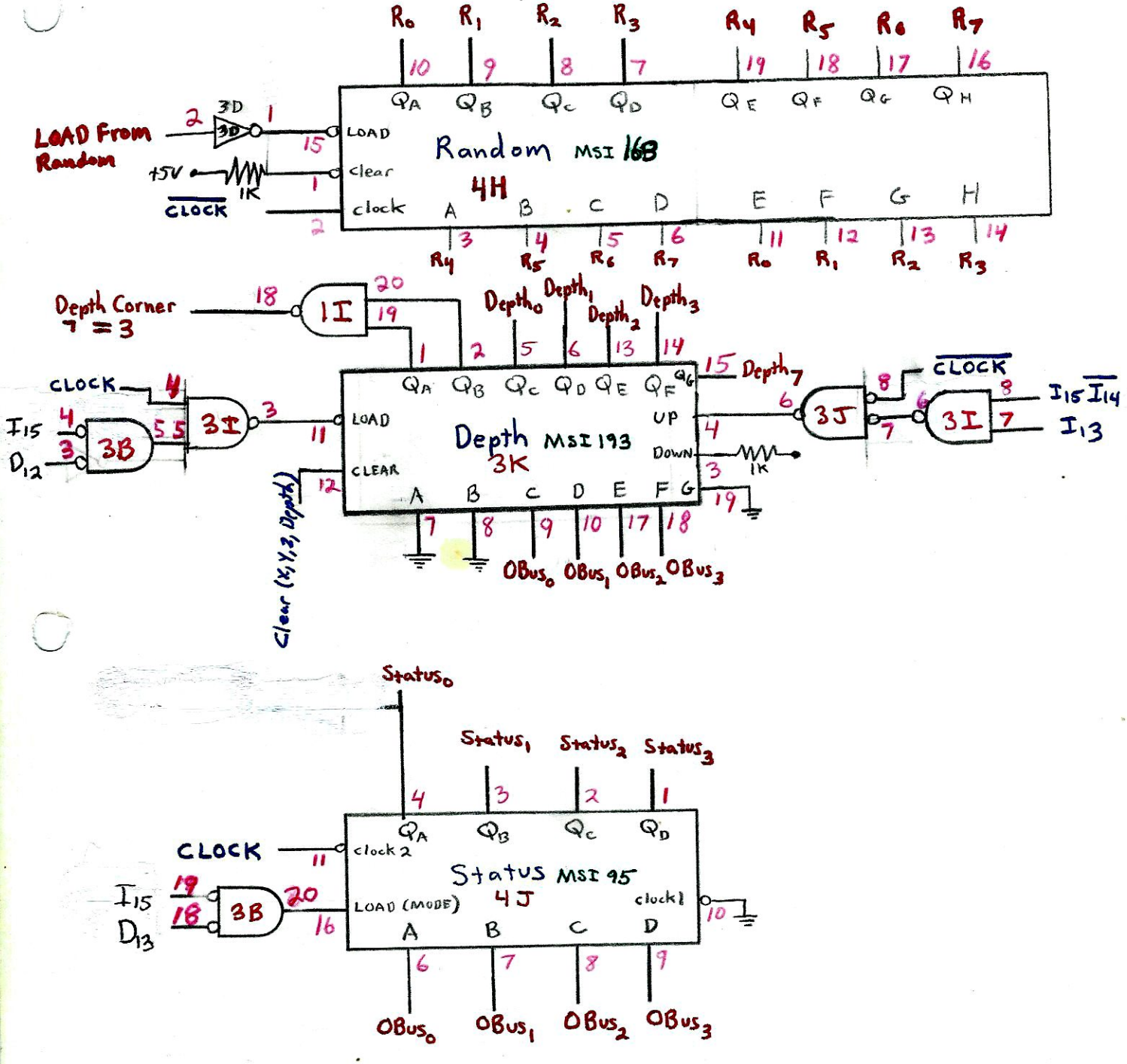

7
6
2D
5
$I_{15}\ \overline{I_{14}}$
I_{15}
$\overline{I_{14}}$
3
2
2D
4
$\overline{I_{15}}\ \overline{I_{14}}$

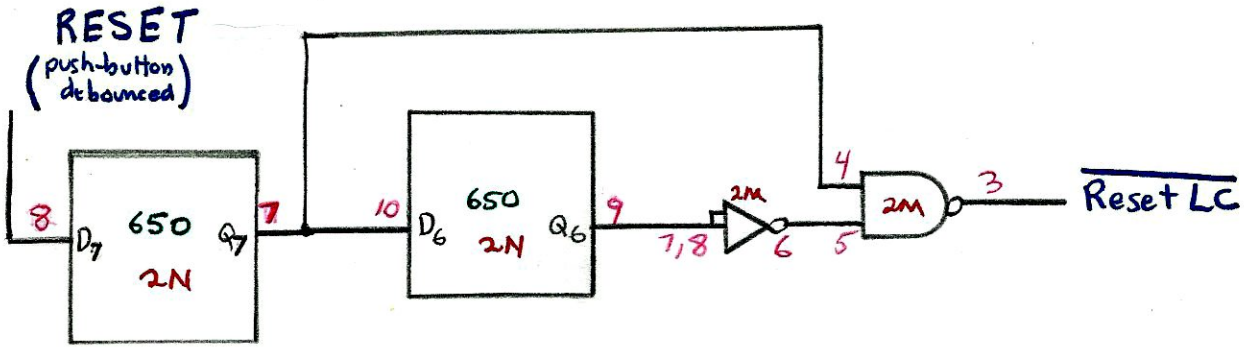
ALU

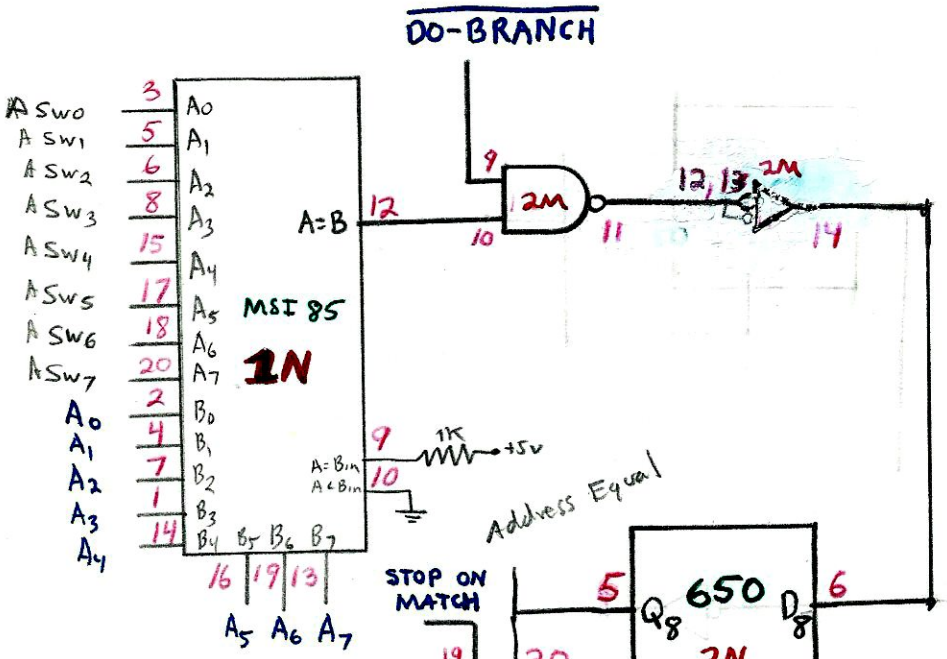

(AC1)
(AC2)
(ABus)
B Selector
MSI 157
1J
ALU
MSI 181
1H
Function
Selector
MSI 157
1G
enable
select
1I
2I
2J
540
OBus0
OBus1
OBus2
OBus3
A=B
Ibus0
Ibus1
Ibus2
Ibus3
+5V
1K


ACI0 ACI1 ACI2 ACI3
4 3 2 1
QA QB QC QD
CLOCK
11
clock2
AC1 4N
MSI 95
I15
14
D1
13
3B
12
16
LOAD (MODE)
clock1
10
A B C D
6 7 8 9
OBus0 OBus1 OBus2 OBus3

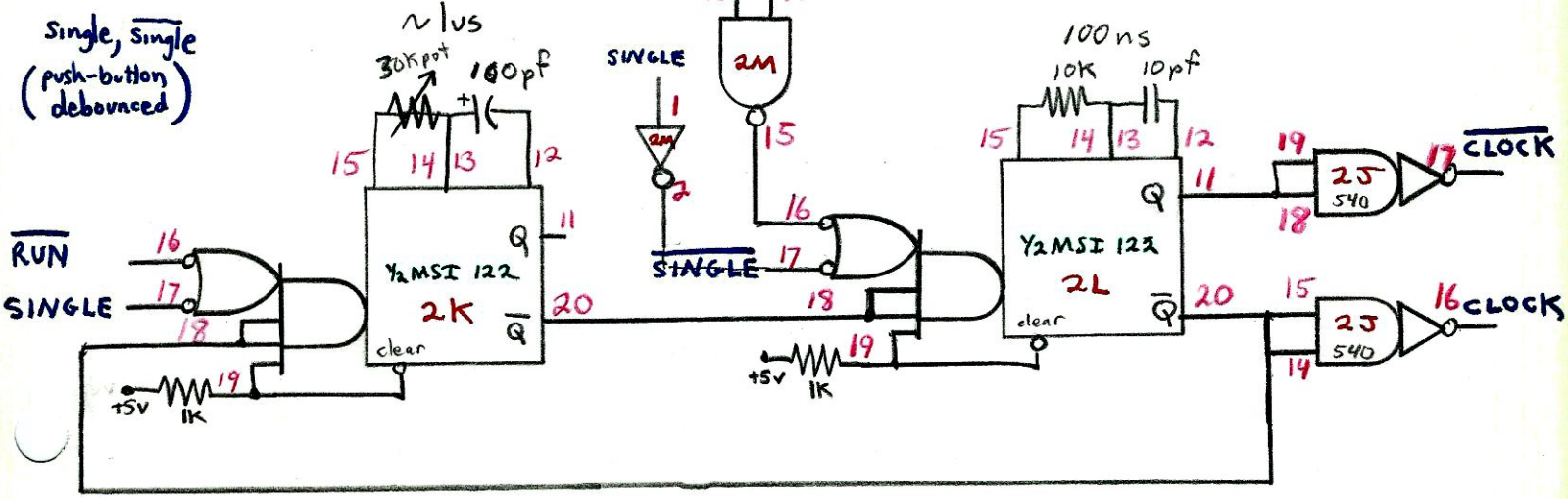

AC20 AC21 AC32 AC23
4 3 2 1
QA QB QC QD
CLOCK
11
clock2
AC2
MSI 95
3M
I15
16
D2
15
3B
17
16
LOAD (MODE)
clock1
10
A B C D
6 7 8 9
OBus0 OBus1 OBus2 OBus3

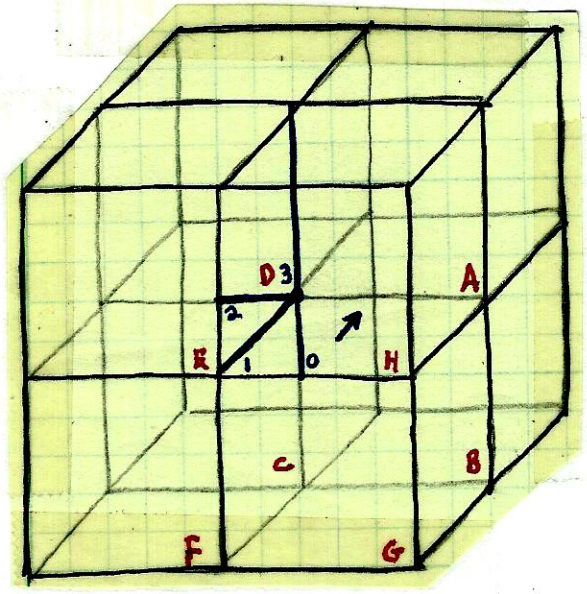

R0 R1 R2 R3 R4 R5 R6 R7
LOAD From Random
3D
+5V
1K
CLOCK
LOAD
clear
clock
Random MSI 16B
4H
QA QB QC QD QE QF QG QH
A B C D E F G H
Depth Corner 7 = 3
1I
Depth0 Depth1 Depth2 Depth3
Depth7
CLOCK
3B
3I
3J
I15 I14 I13 D12
LOAD
CLEAR
UP
DOWN
Depth MSI 193
3K
Clear (X), Y, Z, Depth)
OBus0 OBus1 OBus2 OBus3
Status0 Status1 Status2 Status3
clock 2
clock1
LOAD (MODE)
Status MSI 95
4J
I15 D13
OBus0 OBus1 OBus2 OBus3

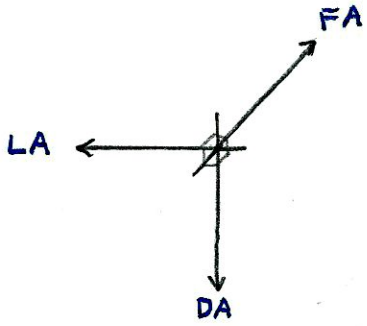


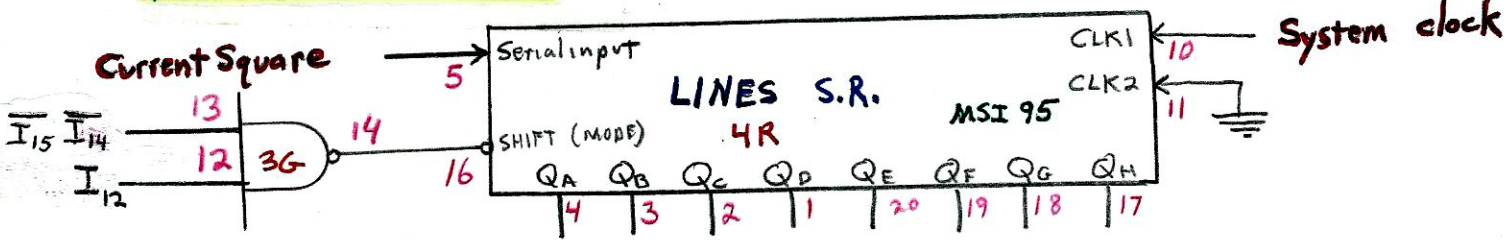


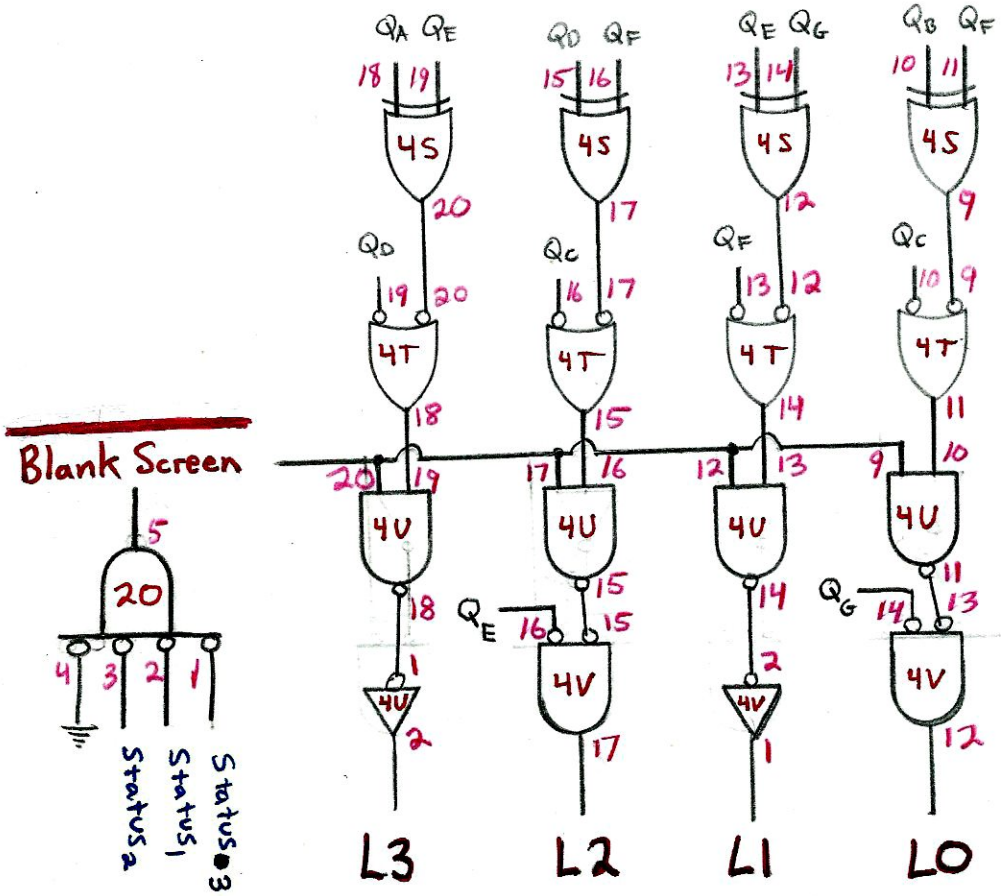
STOP ON MATCH, RUN
(Debounced switches)



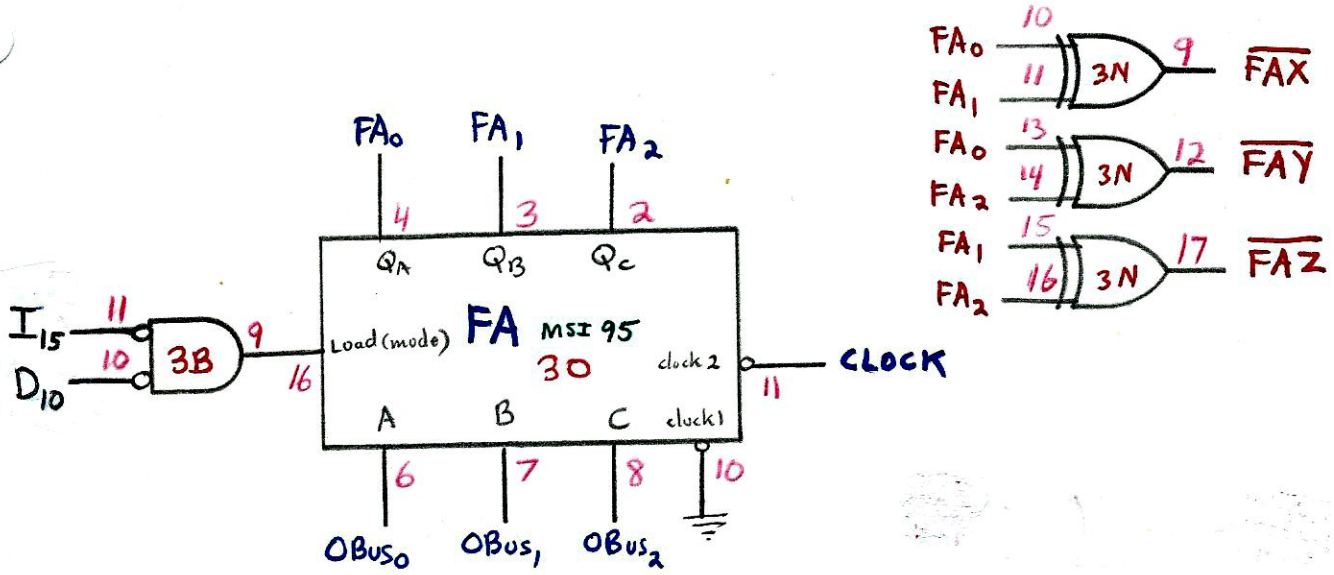




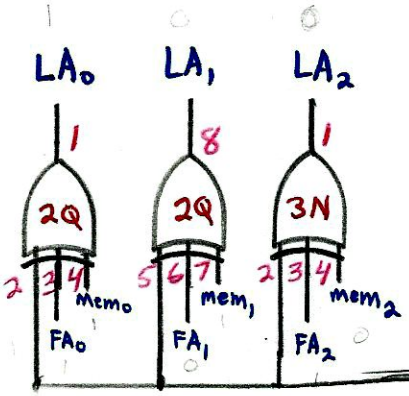


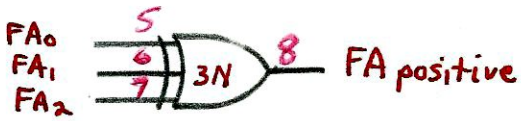


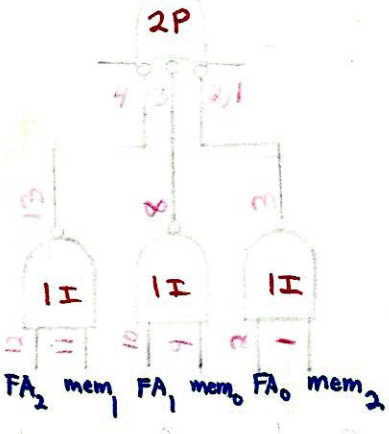
$$L_0 = \overline{C}\,(\overline{C} + \overline{B \oplus F})$$
$$L_1 = \overline{F} + \overline{E \oplus G}$$
$$L_2 = \overline{E}\,(\overline{C} + \overline{D \oplus F})$$
$$L_3 = \overline{D} + \overline{A \oplus E}$$

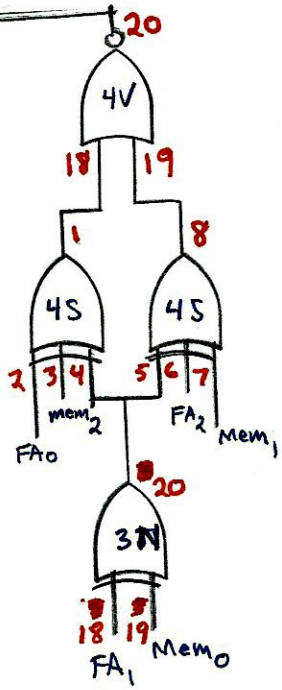

FA0
FA1
FA2
4
3
2
QA
QB
QC
I15
D10
11
10
3B
9
16
Load (mode)
FA MSI 95
30
clock 2
clock 1
CLOCK
11
A
B
C
6
7
8
10
OBus0
OBus1
OBus2

FA0
FA1
10
11
3N
9
FAX
FA0
FA2
13
14
3N
12
FAY
FA1
FA2
15
16
3N
17
FAZ


LA0
LA1
LA2
1
8
1
2Q
2Q
3N
2 3 4
5 6 7
2 3 4
mem0
mem1
mem2
FA0
FA1
FA2

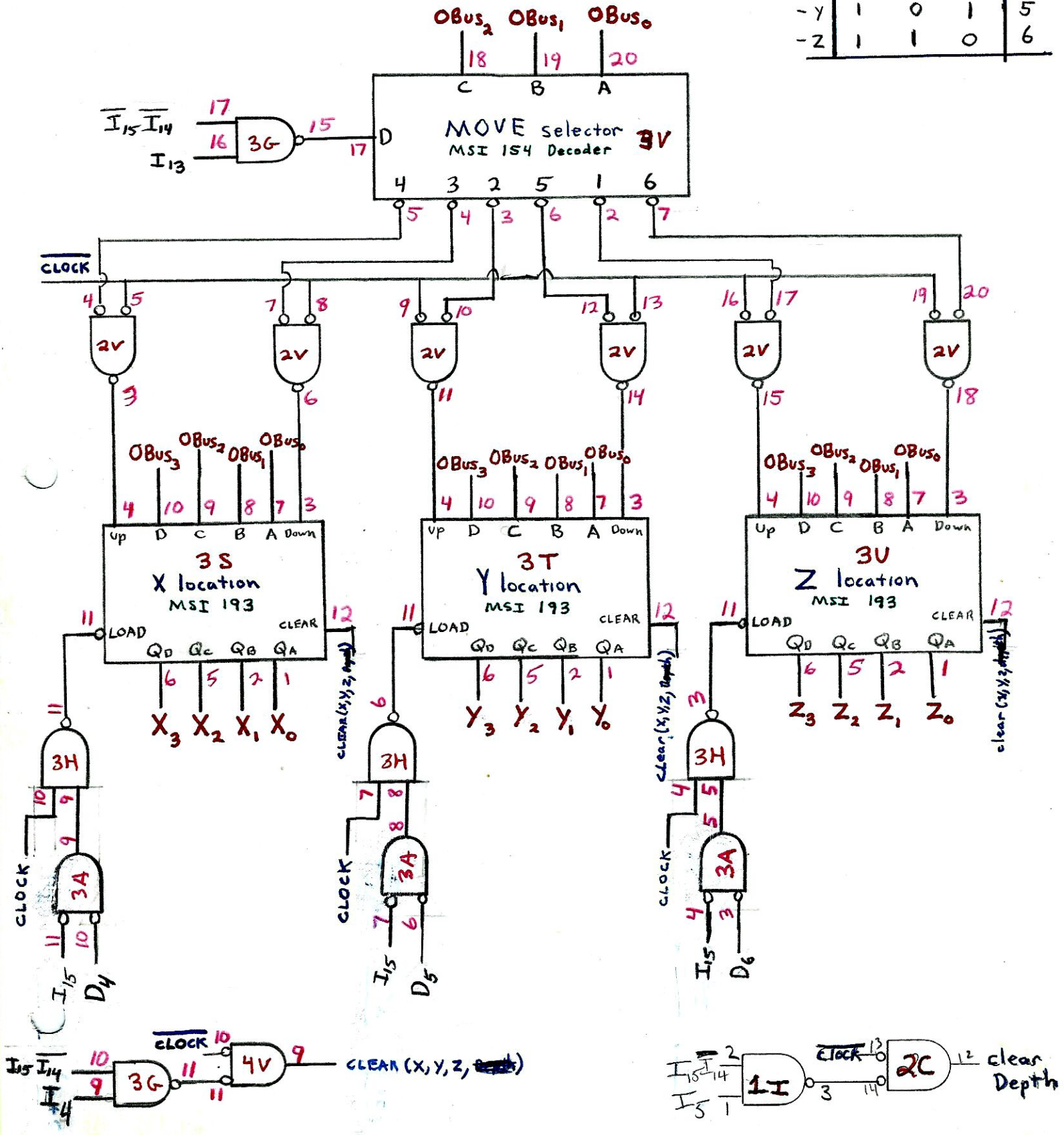

FA0
FA1
FA2
5
6
7
3N
8
FA positive

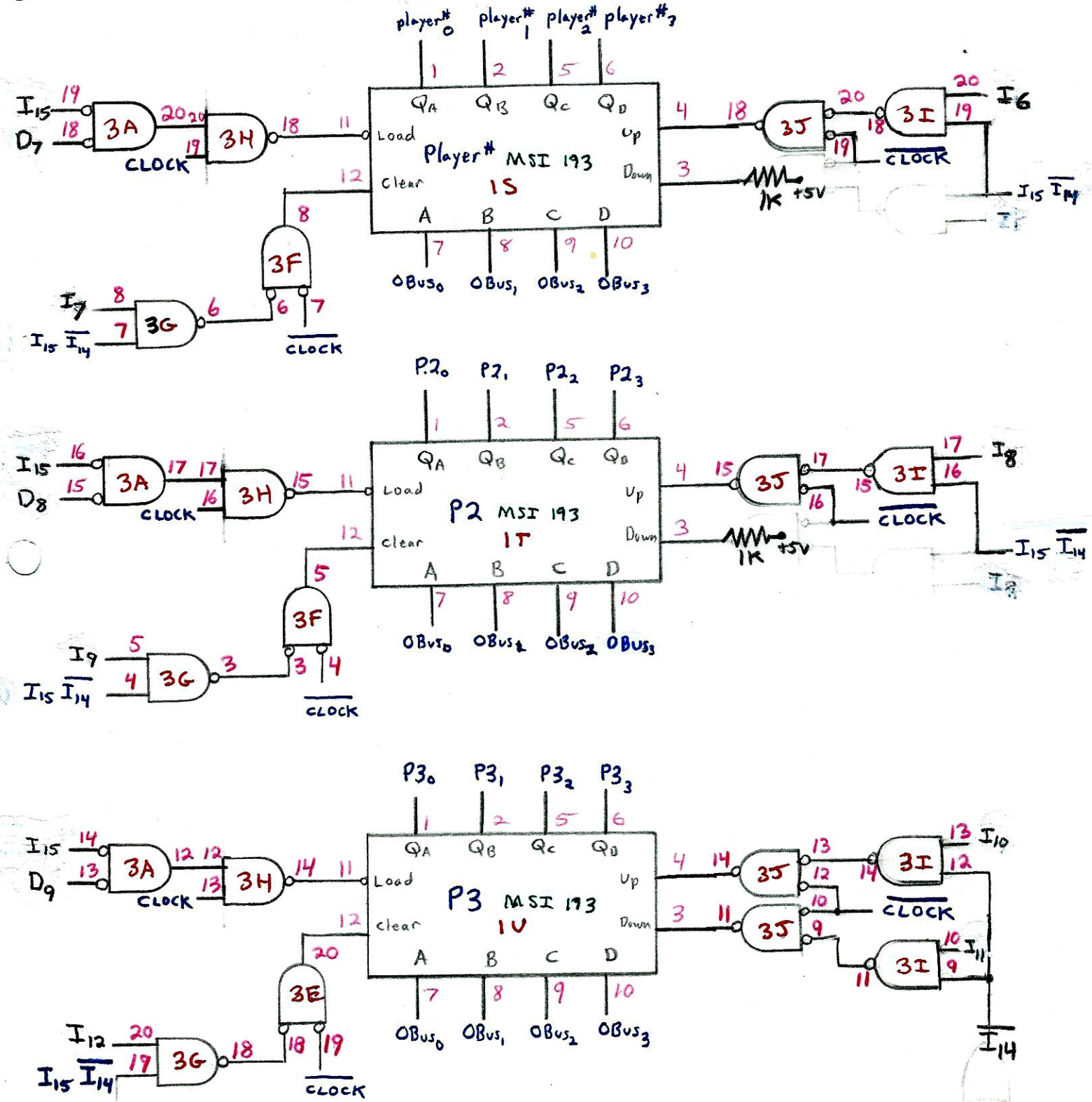

2P
1I
1I
1I
FA2
mem1
FA1
mem0
FA0
mem2

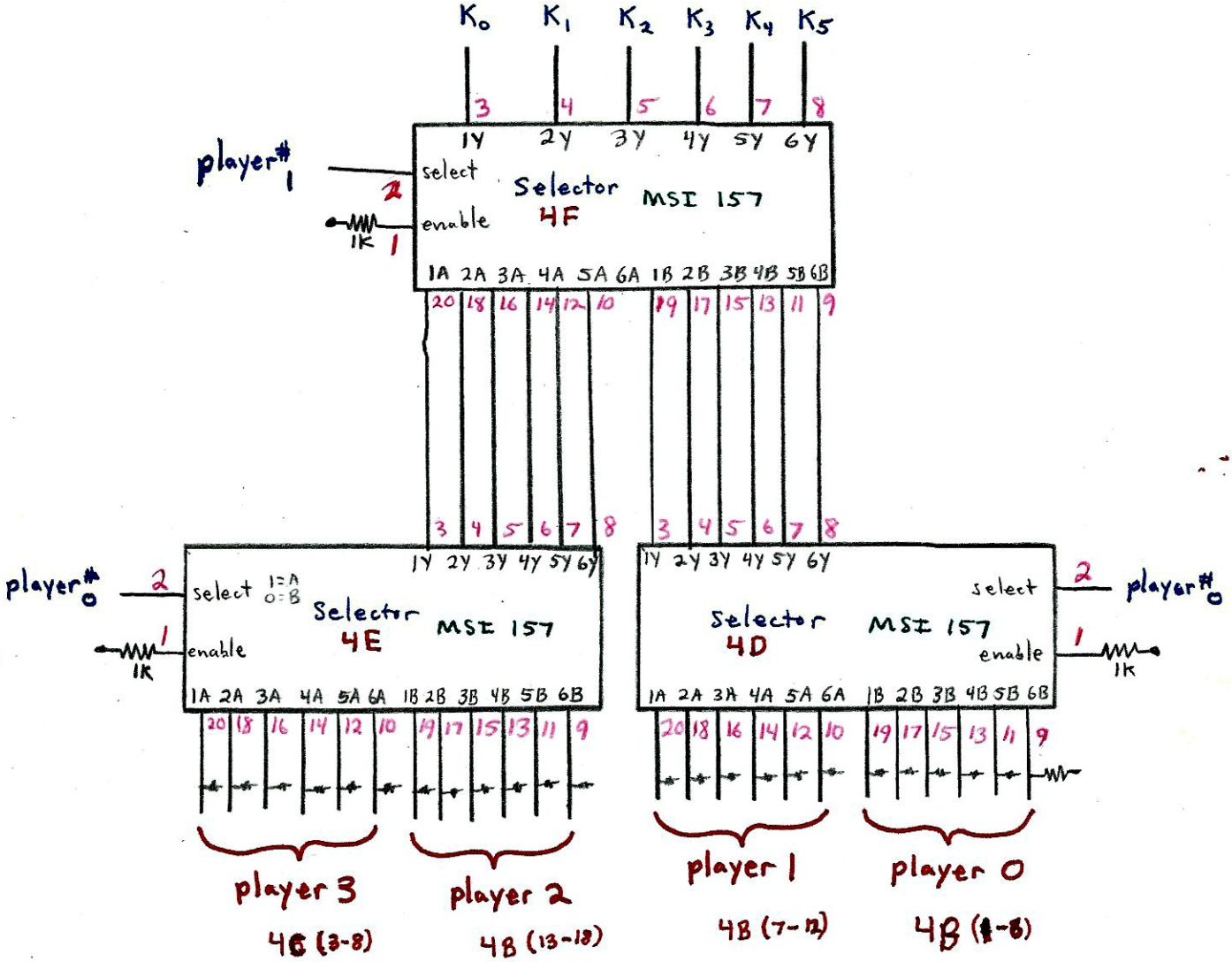

20
4V
18
19
1
8
4S
4S
2 3 4
5 6 7
mem2
FA2
mem1
FA0
20
3N
18
19
FA1
Mem0

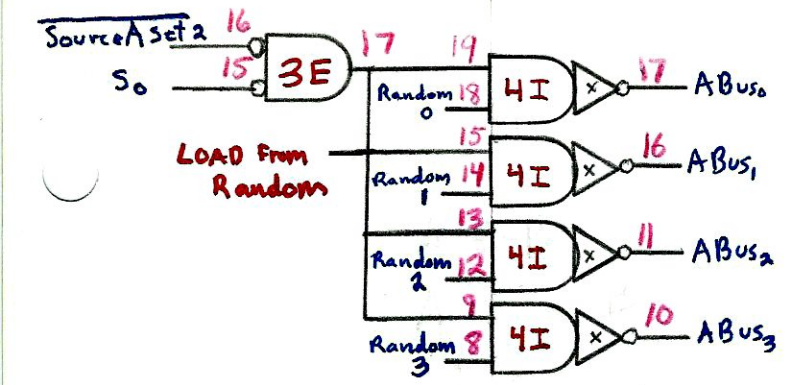
AXIS ENCODING :

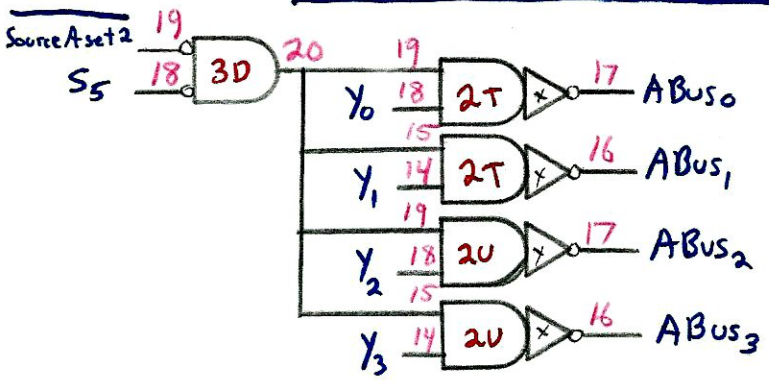
	C $OBus_2$	B $OBus_1$	A $OBus_0$	
x	1	0	0	4
y	0	1	0	2
z	0	0	1	1
-x	0	1	1	3
-y	1	0	1	5
-z	1	1	0	6

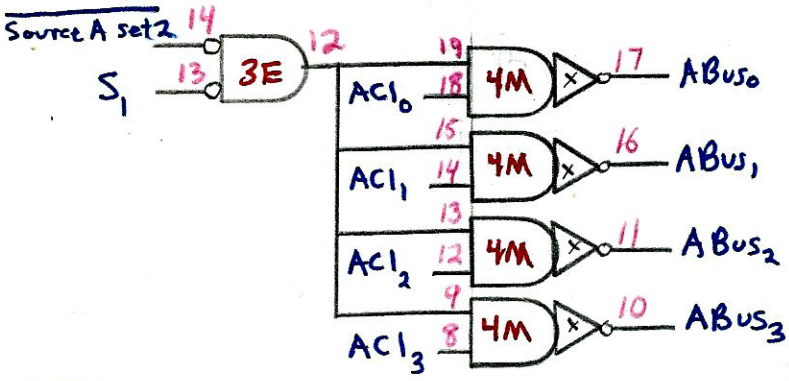


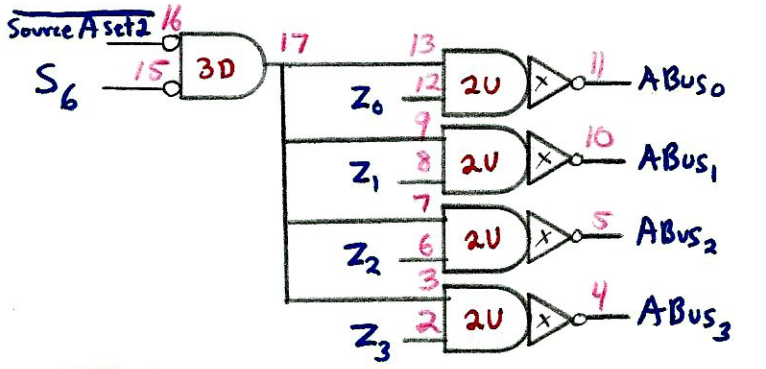

player#0
player#1
player#2
player#3
Player# MSI 193
1S
Load
Clear
Up
Down
A
B
C
D
QA
QB
QC
QD
OBus0
OBus1
OBus2
OBus3
3A
3H
3F
3G
3J
3I
CLOCK
1K
+5V
I15
D7
I9
I14
I6
P20
P21
P22
P23
P2 MSI 193
1T
D8
I8
P30
P31
P32
P33
P3 MSI 193
1U
D9
3E
I12
I10
I11

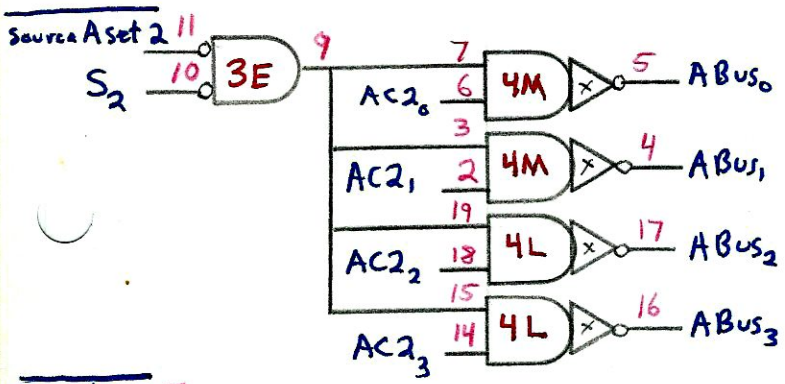

K0 K1 K2 K3 K4 K5
player# 1
select
enable
1K
Selector MSI 157
4F
player# 0
Selector MSI 157
4E
Selector MSI 157
4D
player 3
4E (3-8)
player 2
4B (13-18)
player 1
4B (7-12)
player 0
4B (1-6)

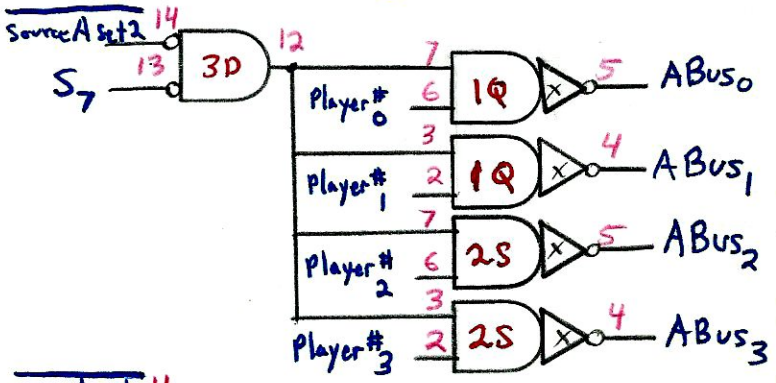

SourceA set 2
S0
3E
LOAD From Random
Random 0
Random 1
Random 2
Random 3
4I
ABUS0
ABUS1
ABUS2
ABUS3

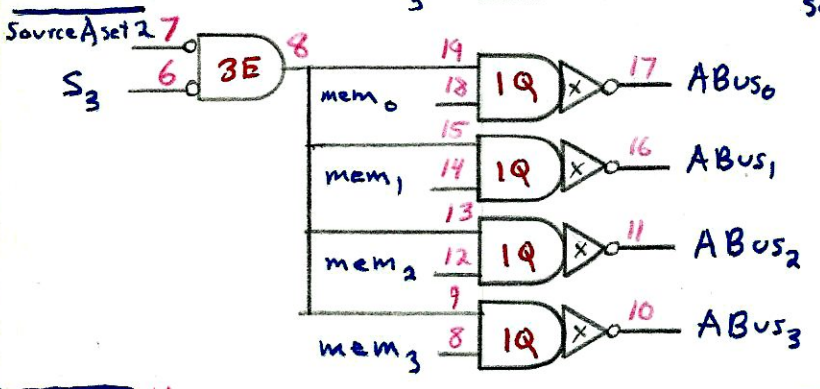

SourceA set 2
S5
3D
Y0
Y1
Y2
Y3
2T
2U
ABUS0
ABUS1
ABUS2
ABUS3

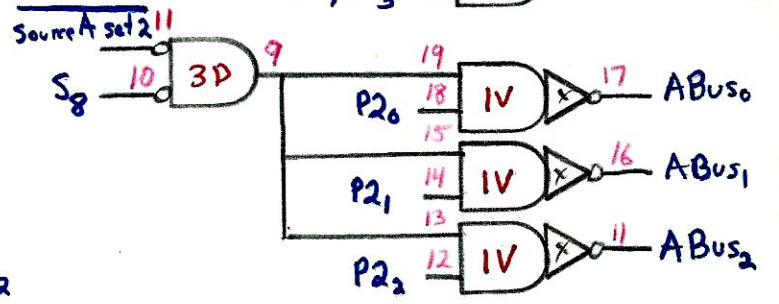

SourceA set 2
S1
3E
ACL0
ACL1
ACL2
ACL3
4M
ABUS0
ABUS1
ABUS2
ABUS3

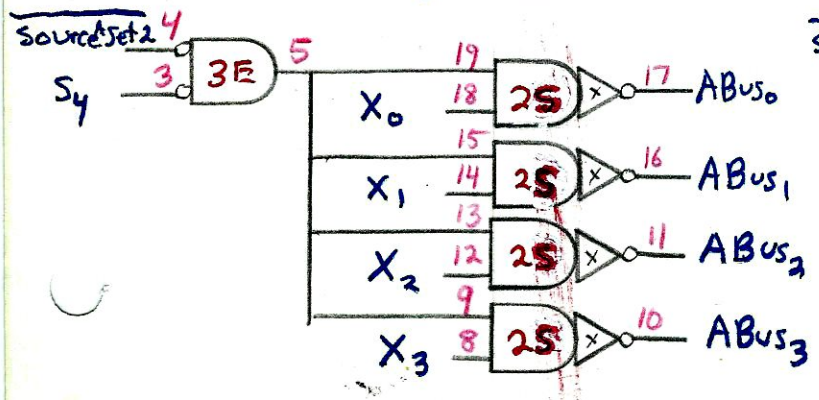

SourceA set 2
S6
3D
Z0
Z1
Z2
Z3
2U
ABUS0
ABUS1
ABUS2
ABUS3

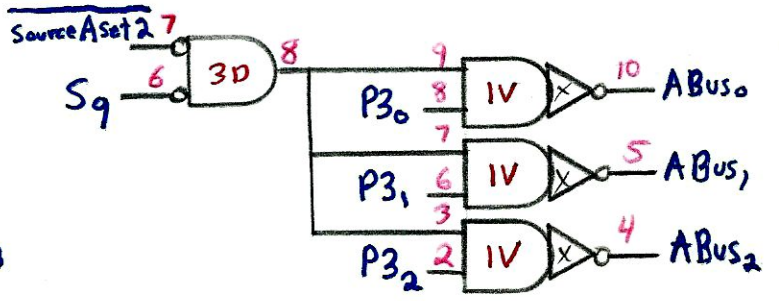

SourceA set 2
S2
3E
AC20
AC21
AC22
AC23
4M
4L
ABUS0
ABUS1
ABUS2
ABUS3

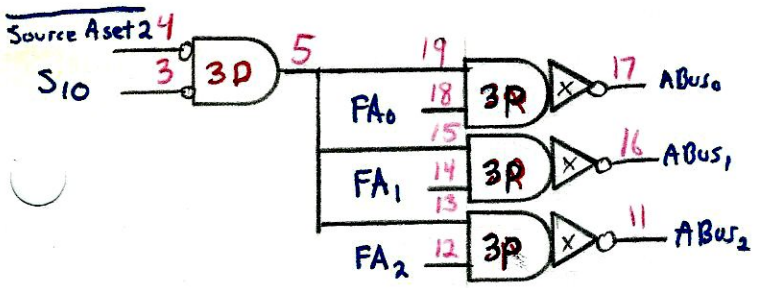

SourceA set 2
S7
3D
Player# 0
Player# 1
Player# 2
Player# 3
1Q
2S
ABUS0
ABUS1
ABUS2
ABUS3

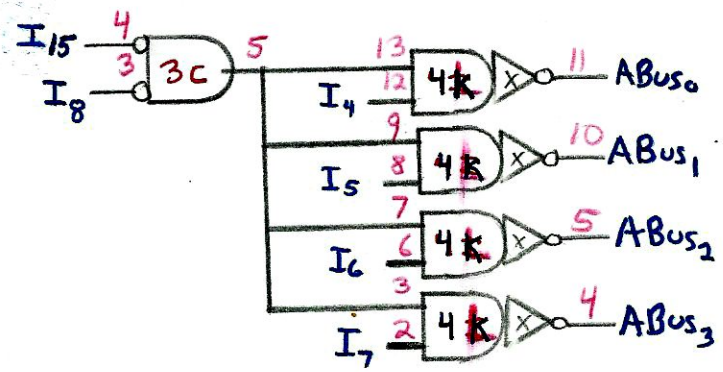

SourceA set 2
S3
3E
mem0
mem1
mem2
mem3
1Q
ABUS0
ABUS1
ABUS2
ABUS3

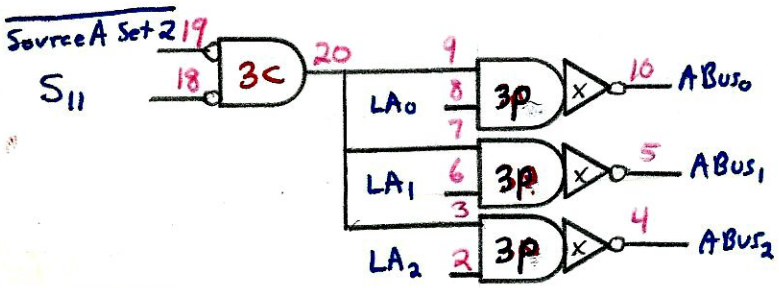

SourceA set 2
S8
3D
P20
P21
P22
1V
ABUS0
ABUS1
ABUS2

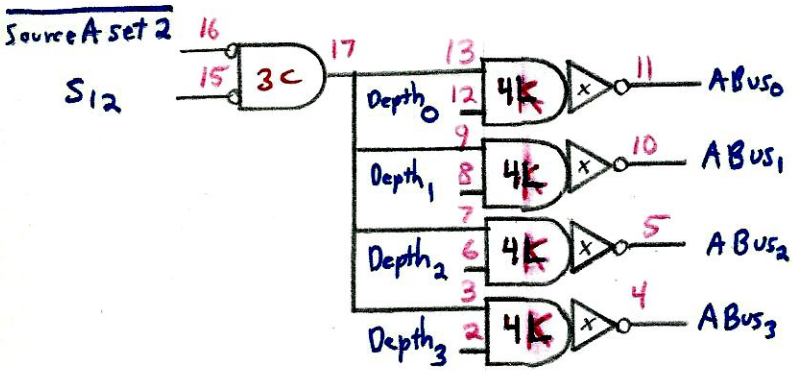

SourceA set 2
S4
3E
X0
X1
X2
X3
2S
ABUS0
ABUS1
ABUS2
ABUS3

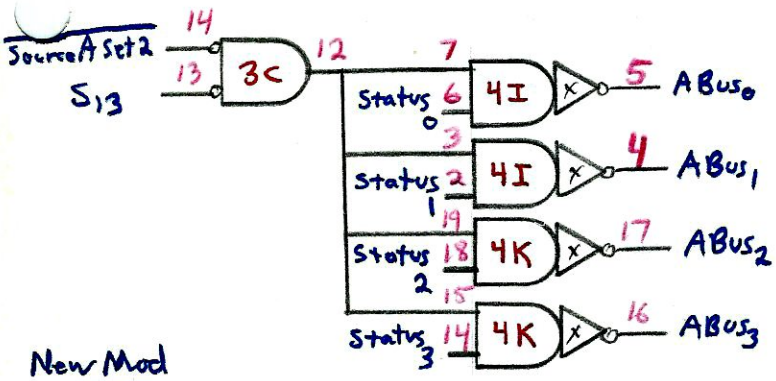

SourceA set 2
S9
3D
P30
P31
P32
1V
ABUS0
ABUS1
ABUS2



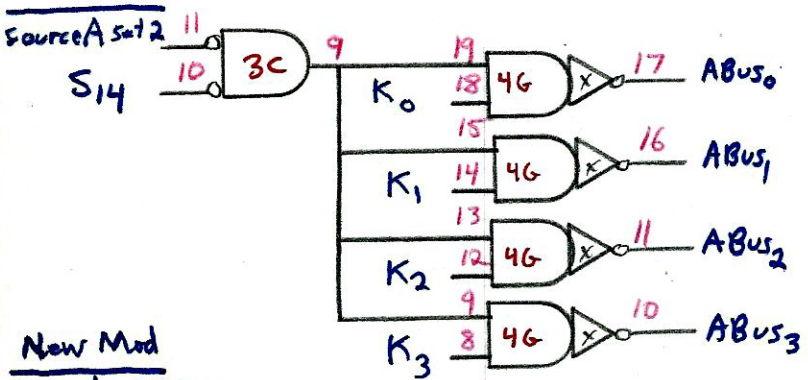




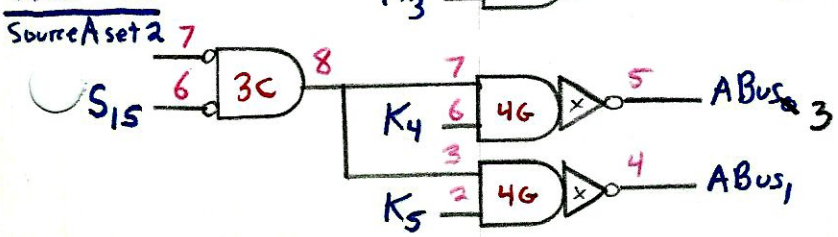


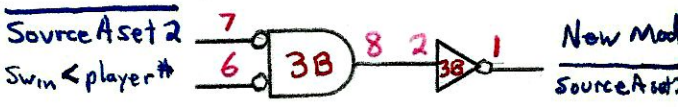


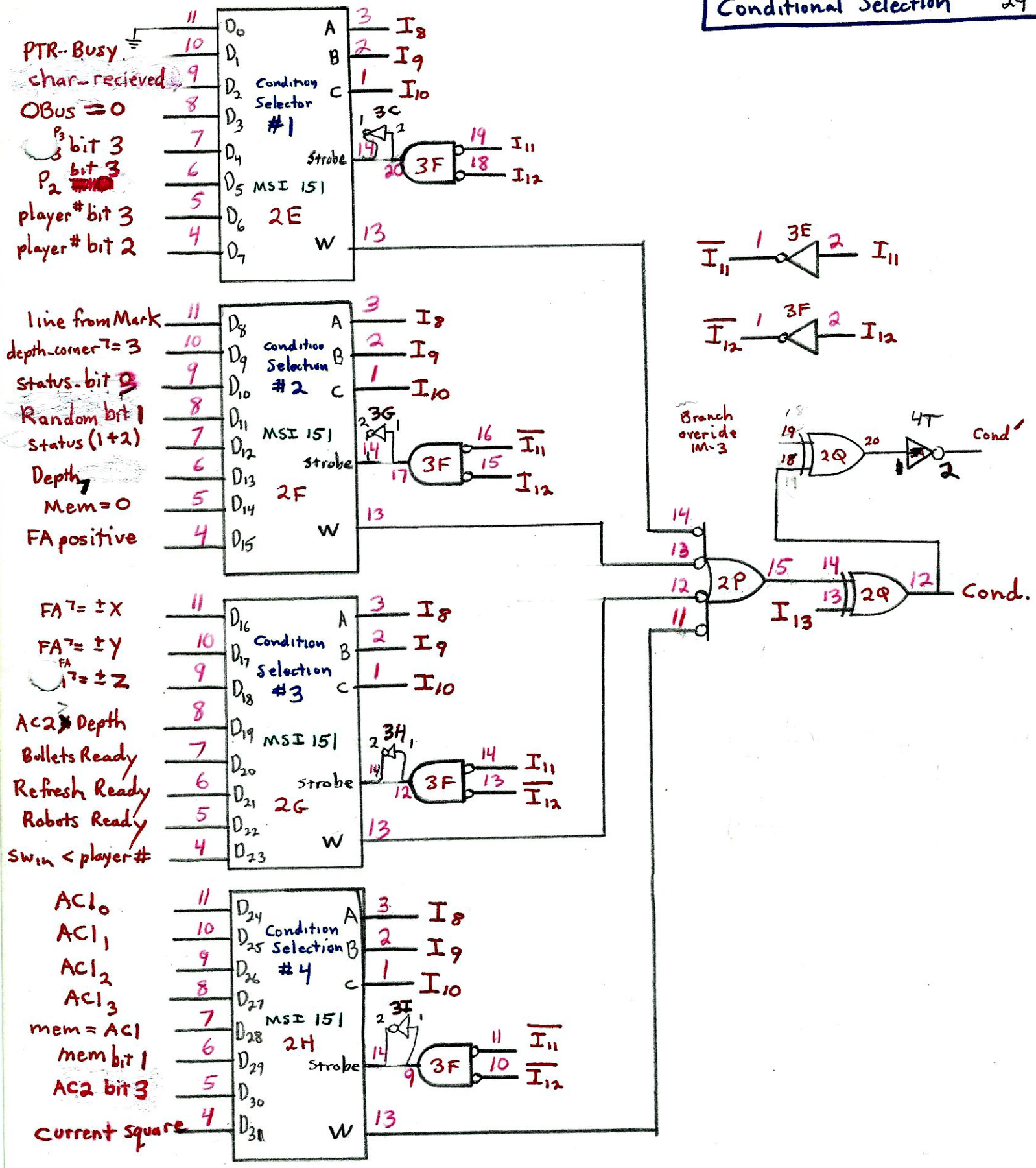
New Mod

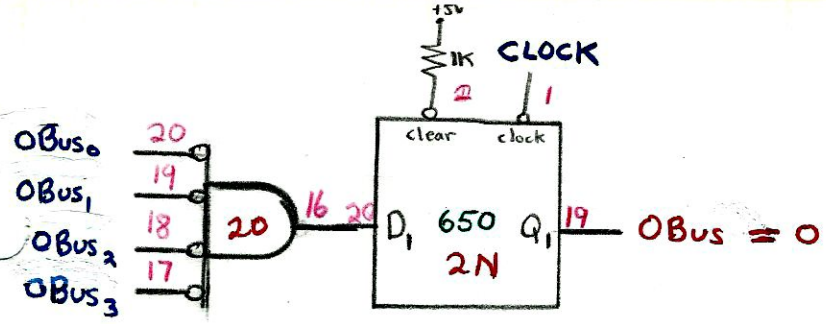


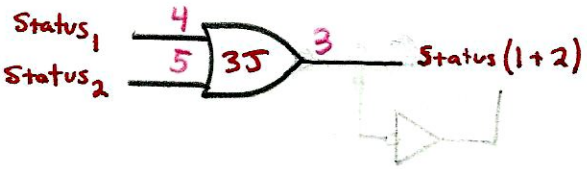
New Mod

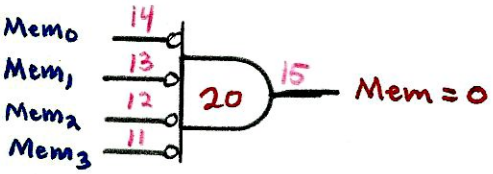


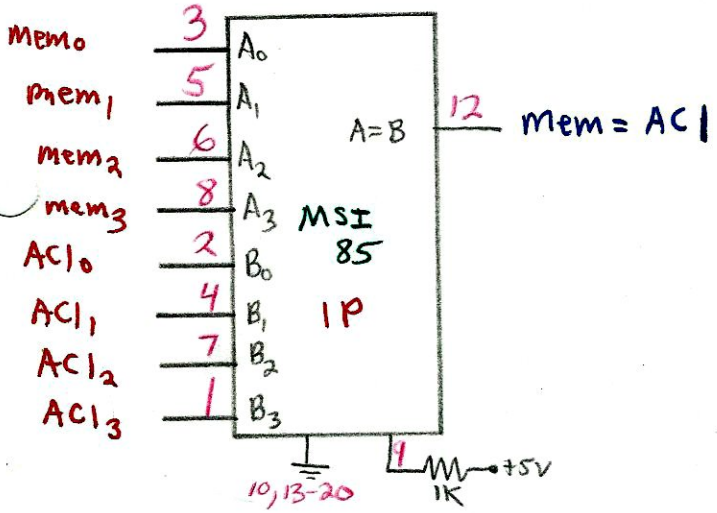


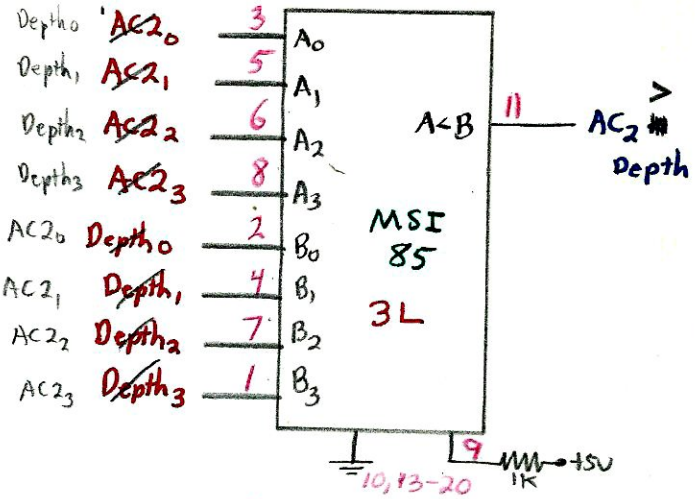

PTR-Busy
char_recieved
OBus = 0
P3 bit 3
P2 bit 3
player# bit 3
player# bit 2
Condition Selector #1
MSI 151
2E
Strobe
3C
3F
line from Mark
depth_corner[7] = 3
status_bit 3
Random bit 1
status (1+2)
Depth 7
Mem = 0
FA positive
Condition Selection #2
MSI 151
2F
3G
FA[7] = ±X
FA[7] = ±Y
FA[7] = ±Z
AC2 > Depth
Bullets Ready
Refresh Ready
Robots Ready
SWin < player#
Condition Selection #3
MSI 151
2G
3H
ACl0
ACl1
ACl2
ACl3
mem = ACl
mem bit 1
AC2 bit 3
current square
Condition Selection #4
MSI 151
2H
3I
Branch overide IM-3
2Q
4T
Cond'
2P
2Q
Cond.
3E
3F

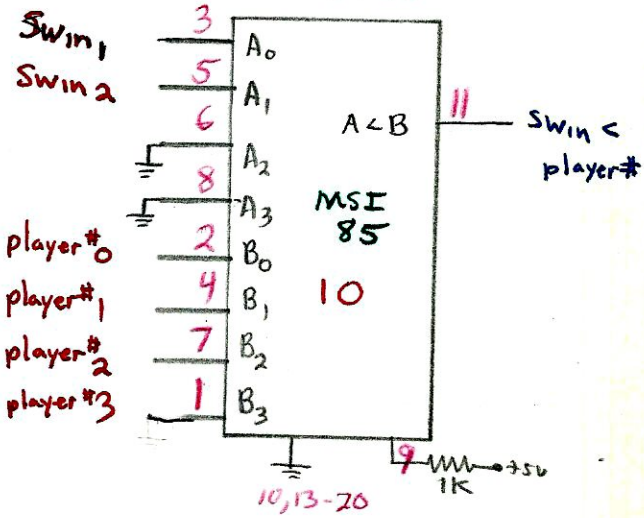

+5V
1K
CLOCK
clear
clock
OBus0
OBus1
OBus2
OBus3
20
650
2N
D1
Q1
OBus = 0

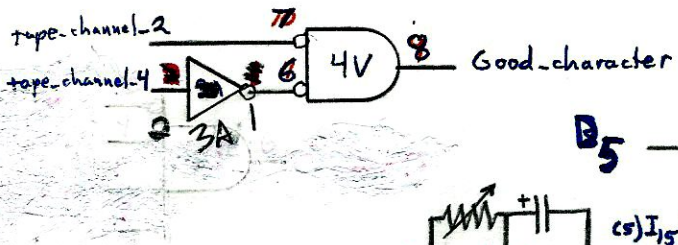

Status1
Status2
35
Status (1+2)

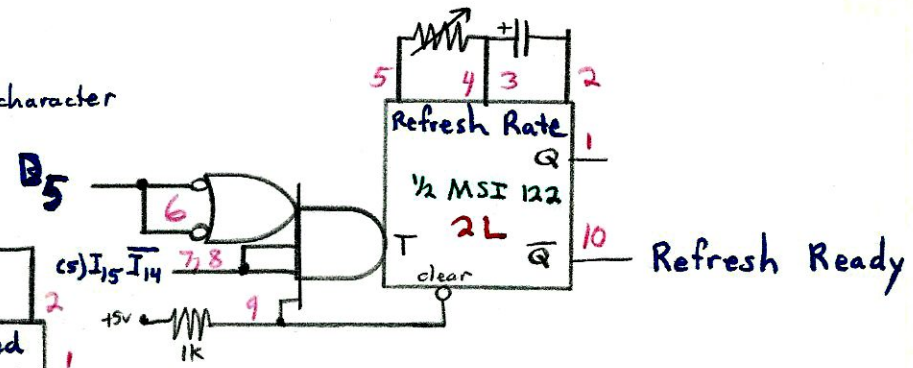

Mem0
Mem1
Mem2
Mem3
20
Mem = 0

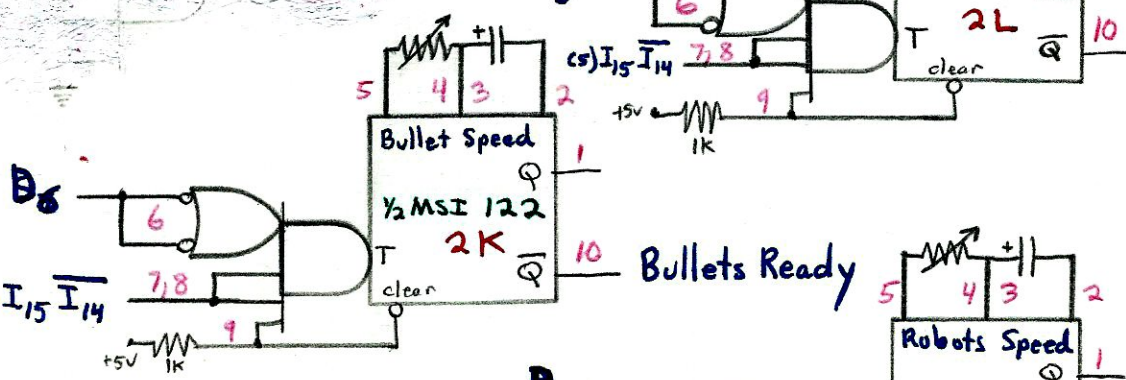

Mem0
mem1
mem2
mem3
AC10
AC11
AC12
AC13
A=B
MSI
85
1P
Mem = AC1
10,13-20
+5V
1K

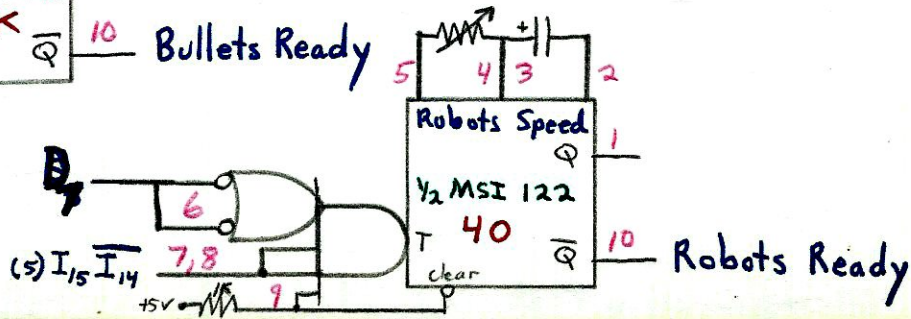

Depth0 AC20
Depth1 AC21
Depth2 AC22
Depth3 AC23
AC20 Depth0
AC21 Depth1
AC22 Depth2
AC23 Depth3
A<B
MSI
85
3L
AC2 > Depth
10,13-20
+5V
1K

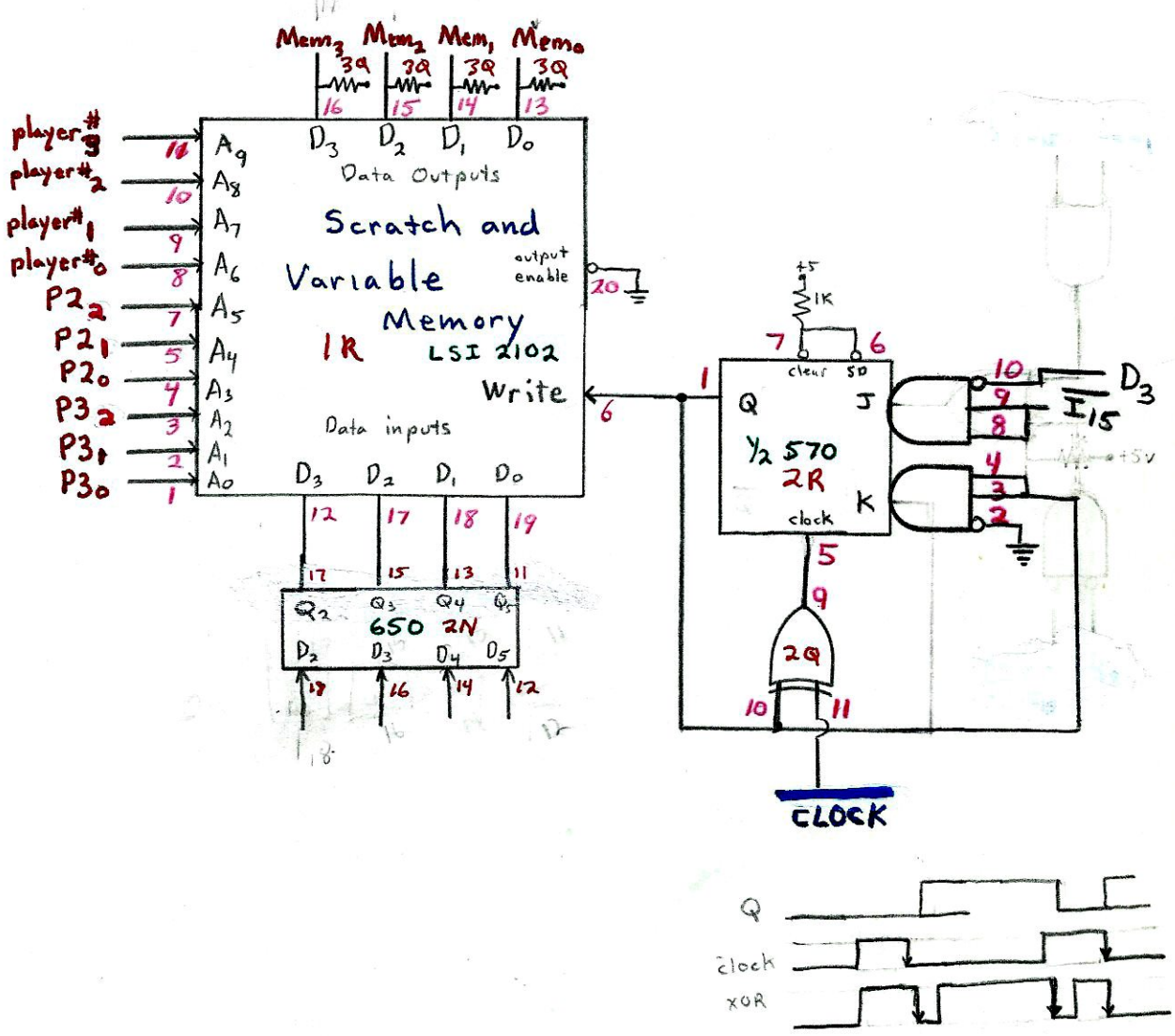

Swin1
Swin2
player#0
player#1
player#2
player#3
A<B
MSI
85
10
Swin < player#
10,13-20
+5V
1K

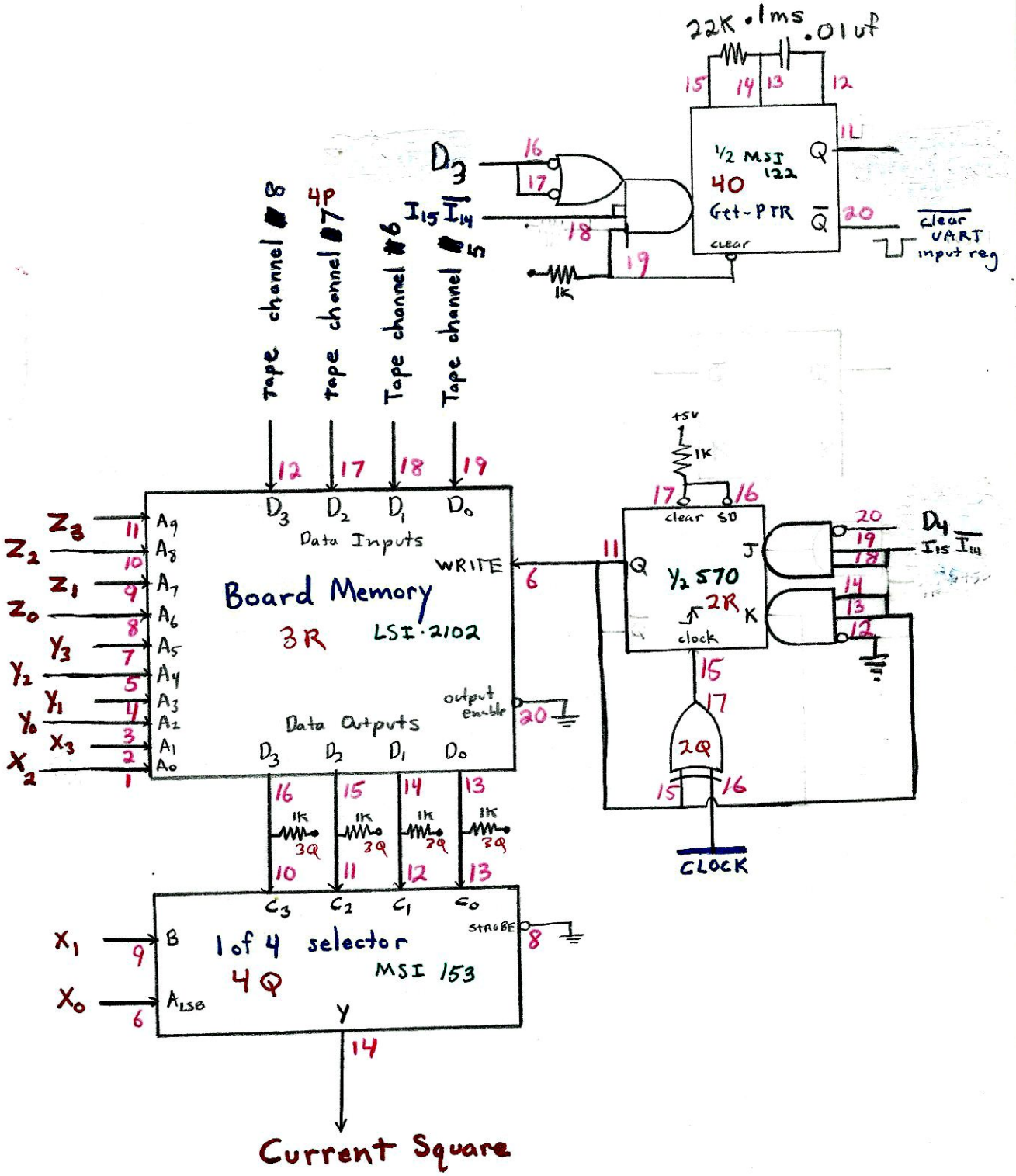

tape_channel_2
tape_channel_4
4V
3A
Good_character


B5
(5) I15 I14
Refresh Rate
½ MSI 122
2L
clear
Refresh Ready
+5V
1K


B6
(5) I15 I14
Bullet Speed
½ MSI 122
2K
clear
Bullets Ready
+5V
1K


B7
(5) I15 I14
Robots Speed
½ MSI 122
40
clear
Robots Ready
+5V


Mem3
Mem2
Mem1
Mem0
39
16
15
14
13
player#3
player#2
player#1
player#0
P22
P21
P20
P32
P31
P30
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
D3
D2
D1
D0
Data Outputs
Scratch and
Variable
Memory
1K
LSI 2102
output enable
20
Write
Data inputs
12
17
18
19
Q2
Q3
Q4
Q5
650 2N
+5
1K
7
6
clear
SD
1
Q
J
½ 570
2K
K
clock
5
9
2Q
10
11
CLOCK
D3
I15
+5V
Q
clock
XOR


22K .1ms .01uf
1/2 MSI 122
40
Get-PTR
clear
clear UART input reg
D3
I15 I14
1K
tape channel #8
tape channel #7
Tape channel #6
Tape channel #5
4P
Board Memory
3R
LSI-2102
Data Inputs
Data Outputs
WRITE
output enable
Z3
Z2
Z1
Z0
Y3
Y2
Y1
Y0
X3
X2
1 of 4 selector
4Q
MSI 153
X1
X0
STROBE
Current Square
+5V
clear SD
Y2 570
2R
clock
J
K
D4
I15 I14
2Q
CLOCK

Reset, clock, and Address Stop

reset pushbutton: 1M-1 to 2N-8
single/cycle pushbutton: 1M-2 to 2K-17, 2M-1
Run switch: 1M-12 to 2M-16
$\overline{\text{Run}}$ switch: 1M-11 to 2K-16
match switch: 1M-10 to 2M-19

2N-7 to 2N-10, 2M-4
2N-9 to 2M-7, 2M-8
2M-6 to 2M-5
1N-12 to 2M-10
2M-11 to 2M-12, 2M-13
2M-14 to 2N-6
2N-5 to 2M-20, 1K-12
2M-18 to 2M-17
2M-15 to 2L-16
2M-2 to 2L-17
2K-20 to 2L-18
2L-11 to 2J-18, 2J-19
2L-20 to 2J-14, 2J-15, 2K-18

Aswitches: Asw_0: 1M-13 to 1N-3
Asw_1: 1M-14 to 1N-5
1M-15 to 1N-6
1M-16 to 1N-8
1M-17 to 1N-15
1M-18 to 1N-17
1M-19 to 1N-18
1M-20 to 1N-20

$\overline{\text{Reset}}$LC: 2M-3 to 1C-1

$\overline{\text{clock}}$: 2J-17 to 1C-2, 3F-7, 3F-4, 3E-19, 3J-19, 3J-16, 3J-12, 3J-10, 4H-2, 3J-8, 2Q-11, 2Q-16, 2V-5, 2V-8, 2V-9, 2V-13, 2V-16, 2V-19, 4V-10,

---

clock: 2J-16 to 1A-11, 1E-11, 3H-19, 3H-16, 3H-13, 3I-4, 4J-11, 4N-11, 3M-11, 3Q-11, 3H-10, 3H-7, 3H-4, 4R-10, 2N-1, 2D-19

10K resistor ~~2L-15 to 2L-14~~
10pf capacitor ~~2L-13 to 2L-12~~
30K pot. ~~2K-15 to 2K-14~~
100pf capacitor ~~2K-13 to 2K-12~~

Logic 1 sources: 1A-1, 1Q-1, 2I-1, 2I-2, 2I-3, 2I-4, 3R-1, 2S-1, 2T-1, 2U-1, 4M-1, 4L-1, 4K-1, 4I-1, 4G-1

Logic 1 Destinations: 2R-19, 2L-19, 1N-9, 4H-15, 4H-1, 3H-3, 2R-7, 2R-6, 2R-17, 2R-16, 4Q-19, 4F-1, 4E-1, 4D-1, 2N-2, 3L-9, 1O-9, 1P-9, 2K-9, 2L-9, 4O-9, 1S-3, 1T-3, 1J-1, 1G-1, 1G-18

Ground Destinations: 1D-12, 1B-12, 1E-10, 1A-10, 1N-10, ——, 4N-10, 3M-10, 3K-7, 3K-8, 3K-19, 4J-10, 2E-11, 2R-2, 1R-20, 3R-20, 4Q-8, 2R-12, 3O-10, 2Q-4, 4R-11, 3L-10, 3L-(13-20), 1O-10, 1O-(13-20), 1P-10, 1P-(13-20), 1O-6, 1O-8, 1G-16, 1G-14, 1G-12, 1G-10

LC, Program Memory and Branch control

memory lines:

1D-20	to 1E-15	1B-20	to 1A-15
1D-19	to 1E-14	1B-19	to 1A-14
1D-18	to 1E-13	1B-18	to 1A-13
1D-17	to 1E-12	1B-17	to 1A-12
1D-16	to 1E-9	1B-16	to 1A-9
1D-15	to 1E-8	1B-15	to 1A-8
1D-14	to 1E-7	1B-14	to 1A-7
1D-13	to 1E-6	1B-13	to 1A-6

A_0: 1C-10 to 1D-11, 1B-11, 1K-13, 1N-2

A_1: 1C-9 to 1D-10, 1B-10, 1K-14, 1N-4

A_2: 1C-8 to 1D-9, 1B-9, 1K-15, 1N-7

A_3: 1C-7 to 1D-8, 1B-8, 1K-16, 1N-1

A_4: 1C-19 to 1D-7, 1B-7, 1K-17, 1N-14

A_5: 1C-18 to 1D-6, 1B-6, 1K-18, 1N-16

A_6: 1C-17 to 1D-5, 1B-5, 1K-19, 1N-19

A_7: 1C-16 to 1D-4, 1B-4, 1K-20, 1N-13

I_0: 1A-4 to 1F-3, 1C-3, 2A-20

I_1: 1A-3 to 1F-4, 1C-4, 2A-19

I_2: 1A-2 to 1F-5, 1C-5, 2A-18

I_3: 1A-1 to 1F-6, 1C-6, 2A-17

I_4: 1A-20 to 1F-7, 1C-11, 4K-12, 2B-20

I_5: 1A-19 to 1F-8, 1C-12, 3G-9, 4K-8, 2B-19

I_6: 1A-18 to 1F-9, 1C-13, 3I-20, 4K-6, 2B-18

I_7: 1A-17 to 1F-10, 1C-14, 3G-8, 4K-2, 2B-17

I_8: 1E-4 to 1F-11, 3I-17, 2F-3, 2F-3, 2G-3, 2H-3, 3C-3, 2D-8

I_9: 1E-3 to 1F-12, 3G-5, 2E-2, 2F-2, 2G-2, 2H-2, 1J-2, 1G-20

I_{10}: 1E-2 to 1F-13, 3I-13, 2E-1, 2F-1, 2G-1, 2H-1, 1G-15, 1G-9

I_{11}: 1E-1 to 1F-14, 3I-10, 3E-2, 3F-19, 3F-14, 1G-11

I_{12}: 1E-20 to 1F-15, 3G-20, 3F-2, 3F-15, 3F-18, 3G-12, 1G-13, 1G-17

I_{13}: 1E-19 to 1F-16, 3I-7, 2Q-13, 3G-16, 1G-19

I_{14}: 1E-18 to 1F-17, 2P-9, 2C-4, 2D-13, 2D-12,

I_{15}: 1E-17 to 1F-18, 2P-10, ——, 2C-2, 2C-3, 2D-3

DoBranch: 2P-6 to 2N-4, 1E-16, 1A-16, 1C-15, 1K-11, 2M-9

Q_7: 2N-3 to 2P-7

-9v to 1B-2, 1D-2

LINES DECODING

3G-14 to 4R-16

Q_A: 4R-4 to 4S-18
Q_B: 4R-3 to 4S-10.
Q_C: 4R-2 to 4T-16, 4I-10
Q_D: 4R-1 to 4S-15, 4T-19
Q_E: 4R-20 to 4S-19, 4S-13, 4V-16
Q_F: 4R-19 to 4S-16, 4S-11, 4T-13
Q_G: 4R-18 to 4S-14, 4V-14

Blank Screen: 20-5 to 4U-20, 4U-17, 4U-12, 4U-9

4S-20 to	4T-20
4S-17 to	4T-17
4S-12 to	4T-12
4S-9 to	4T-9
4T-18 to	4U-19
4T-15 to	4U-16
4T-14 to	4U-13
4T-11 to	4U-10
4U-18 to	4U-1
4U-15 to	4V-15
4U-14 to	4V-2
4U-11 to	4V-13

L0: 4V-12
L1: 4V-1
L2: 4V-17
L3: 4U-2

Conditional Logic

20-16 to 2N-20

OBus=0: 2N-19 to 2E-8
Status(1+2): 3J-3 to 2F-7
Mem=0: 3D-15 to 2F-5
mem=AC1: 1P-12 to 2H-7

AC2<Depth: 3L-11 to 2G-8
Swin<player#: 10-11 to 2G-4, 3B-6
Refresh Ready: 2L-10 to 2G-6
Bullets Speed: 2K-10 to 2G-7
Robots Ready: 40-10 to 2G-5

$Swin_1$: 1A-6 to 10-3
$Swin_2$: 1A-8 to 10-5

Instruction Decoding

$\overline{I_{15}}$: 2C-1 to 2D-14, 2D-15, 2D-9, 2R-8, 2R-9

I_{15_B}: 2D-16 to 3A-11, 3A-7, 3A-4, 3A-19, 3A-16, 3A-14, 3B-14, 3B-16, 3B-4, 3B-19, 3B-11, 3C-4

$\overline{I_{14}}$: 2D-11 to 2D-2, 3I-12, 3I-9, 1G-2

$\overline{I_{15}}\,\overline{I_{14}}$: 2C-5 to 3G-13, 3G-17

$I_{15}\,\overline{I_{14}}$: 2D-4 to 2D-6, 2D-7, 2C-14, 2C-11, 2C-7

$I_{15}\,\overline{I_{14}}$: 2D-5 to 2K-7, 2K-8, 2L-7, 2L-8, 40-7, 40-8, 3G-10, 3G-7, 3G-4, 3G-19, 3E-19, 3I-16, 2R-18, 2R-19, 40-18, 3I-8

Source A set 2: 2B-10 to 3B-7, 3E-14, 3C-16, 3C-19, 3D-4, 3D-7, 3D-11, 3D-14, 3D-16, 3D-19, 3E-4, 3E-7, 3E-11, 3E-14, 3E-16

D_0: 2A-1
D_1: 2A-2 to 2C-13, 3B-13
D_2: 2A-3 to 2C-10, 3B-15
D_3: 2A-4 to 40-16, 40-17, 2R-10

D_4: 2A-5 to 2A-20, 3A-10
D_5: 2A-6 to 2C-6, 3A-6, 2L-6
D_6: 2A-7 to 2K-6, 3A-3,
D_7: 2A-8 to 40-6, 3A-18

D_8: 2A-9 to 3A-15
D_9: 2A-10 to 3A-13
D_{10}: 2A-11 to 3B-10
D_{12}: 2A-13 to 3B-3
D_{13}: 2A-14 to 3B-18

S_0: 2B-1 to 3E-15
S_1: 2B-2 to 3E-13
S_2: 2B-3 to 3E-10
S_3: 2B-4 to 3E-6
S_4: 2B-5 to 3E-3
S_5: 2B-6 to 3D-18

S_6: 2B-7 to 3D-15
S_7: 2B-8 to 3D-13
S_8: 2B-9 to 3D-10
S_9: 2B-10 to 3D-6
S_{10}: 2B-11 to 3D-3
S_{11}: 2B-12 to 3C-18
S_{12}: 2B-13 to 3C-15
S_{13}: 2B-14 to 3C-13
S_{14}: 2B-15 to 3C-10
S_{15}: 2B-16 to 3C-6

2C-12 to	2C-19
2C-9 to	2C-15
2C-20 to	2C-16
2C-17 to	2C-18
2C-8 to	2D-18

ABus$_0$: ~~1H-11~~ from ~~2I-8~~, ~~4I-17~~, ~~4M-17~~, ~~4M-5~~, ~~1Q-17~~, ~~2S-17~~, ~~2T-17~~, ~~2U-11~~, ~~1Q-5~~, ~~1V-17~~, ~~1U-10~~, ~~3P-17~~, ~~3P-10~~, ~~4L-11~~, ~~4I-5~~, ~~4G-17~~, ~~4G-5~~, ~~4K-11~~, ~~1L-1~~

ABus$_1$: ~~1H-9~~ from ~~2I-7~~, ~~4I-16~~, ~~4M-16~~, ~~4M-4~~, ~~1Q-16~~, ~~2S-16~~, ~~2T-16~~, ~~2U-10~~, ~~1Q-4~~, ~~1V-16~~, ~~1V-5~~, ~~3P-16~~, ~~3P-5~~, ~~4L-10~~, ~~4I-4~~, ~~4G-16~~, ~~4G-4~~, ~~4K-10~~, ~~1L-2~~

ABus$_2$: ~~1H-7~~ from ~~2I-6~~, ~~4I-11~~, ~~4M-11~~, ~~4L-17~~, ~~1Q-11~~, ~~2S-11~~, ~~2U-17~~, ~~2U-5~~, ~~2S-5~~, ~~1V-11~~, ~~1V-4~~, ~~3P-11~~, ~~3P-4~~, ~~4L-5~~, ~~4K-17~~, ~~4G-11~~, ~~4K-5~~, ~~1L-3~~

ABus$_3$: ~~1H-5~~ from ~~2I-5~~, ~~4I-10~~, ~~4M-10~~, ~~4L-16~~, ~~1Q-10~~, ~~2S-10~~, ~~2U-16~~, ~~2U-4~~, ~~2S-4~~, ~~4L-4~~, ~~4K-16~~, ~~4G-10~~, ~~4K-4~~, ~~1L-4~~

Conditional selection

~~3F-20 to 2E-14~~
~~3F-17 to 2F-14~~
~~3F-12 to 2G-14~~
~~3F-9 to 2H-14~~

~~2E-13 to 2P-14~~
~~2F-13 to 2P-13~~
~~2G-13 to 2P-12~~
~~2H-13 to 2P-11~~
~~2P-15 to 2Q-14~~

$\overline{I}_{11}$: ~~3E-1 to 3F-16, 3F-11~~
$\overline{I}_{12}$: ~~3F-1 to 3F-13, 3F-10~~

cond: ~~2Q-12 to 2P-8~~

Paper Tape Reader:

PTR-Busy: ~~4P-14 to 2E-10~~
Tape_channel_4: ~~4P-6 to 2E-9~~
tape_channel_5: ~~4P-7 to 3R-12~~
tape_channel_6: ~~4P-8 to 3R-17~~
tape_channel_7: ~~4P-9 to 3R-18~~
tape_channel_8: ~~4P-10 to 3R-19~~
Ext. forward_drive: ~~4P-13 from 40-11~~

Mark Control Lines:

line from Mark to 2F-11
Ready to Mark from 2C-20, ~~1F-2~~
init to Mark from 2D-17

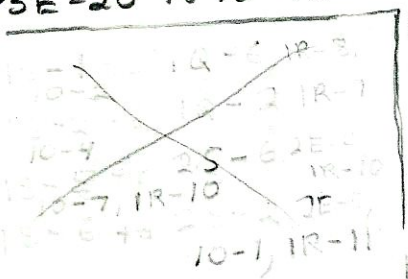
Player #, P2, P3

3A-20 to 3H-20
3H-18 to 1S-11
3G-6 to 3F-6
3F-8 to 1S-12
3A-17 to 3H-17
3H-15 to 1T-11
3G-3 to 3F-3
3F-5 to 1T-12
3A-12 to 3H-12
3H-14 to 1U-11
3G-18 to 3E-18
3E-20 to 1U-12

3I-18 to 3J-20
3J-18 to 1S-4
3I-15 to 3J-17
3J-15 to 1T-4
3I-14 to 3J-13
3J-14 to 1U-4
3I-11 to 3J-9
3J-11 to 1U-3

$P2_0$: 1T-1 to 1V-18, 1R-4, 1K-4
$P2_1$: 1T-2 to 1V-14, 1R-5, 1K-5
$P2_2$: 1T-5 to 1V-12, 1R-7, 1K-6
$P2_3$: 1T-6 to 2E-6

$P3_0$: 1U-1 to 1V-8, 1R-1, 1K-1
$P3_1$: 1U-2 to 1V-6, 1R-2, 1K-2
$P3_2$: 1U-5 to 1V-2, 1R-3, 1K-3
$P3_3$: 1U-6 to 2E-7



player#0: 1S-1 to 1Q-6; 1R-8, 1O-2, 4E-2, 4D-2, 1K-7
player#1: 1S-2 to 1Q-2, 1R-9, 1O-4, 4F-2, 1K-8
player#2: 1S-5 to 2S-6, 2E-4, 1O-7, 1R-10, 1K-9
player#3: 1S-6 to 2S-2, 2E-5, 1O-1, 1R-11, 1K-10

AC1 and AC2

3B-12 to 4N-16
3B-17 to 3M-16

$AC1_0$: 4N-4 to 4M-18, 2H-11, 1A-2, 1J-13, 1L-13
$AC1_1$: 4N-3 to 4M-14, 2H-10, 1A-4, 1J-15, 1L-14
$AC1_2$: 4N-2 to 4M-12, 2H-9, 1P-7, 1J-17, 1L-15
$AC1_3$: 4N-1 to 4M-8, 2H-8, 1A-1, 1J-19, 1L-16

$AC2_0$: 3M-4 to 4M-6, 3L-3, 1J-14, 1L-9
$AC2_1$: 3M-3 to 4M-2, 3L-5, 1J-16, 1L-10
$AC2_2$: 3M-2 to 4L-18, 3L-6, 1J-18, 1L-11
$AC2_3$: 3M-1 to 4L-14, 3L-8, 1J-20, 1L-12

Random, Depth and Status

3B-5 to 3I-5
3I-3 to 3K-11
3I-6 to 3J-7
3J-6 to 3K-4
3B-20 to 4J-16
3K-1 to 1I-19
3K-2 to 1I-20
depth corner 7=3 1I-18 to 2F-10

R_0: ~~4H-10 to 4I-18~~
R_1: ~~4H-9 to 4I-14~~
R_2: ~~4H-8 to 4I-12~~
R_3: ~~4H-7 to 4I-8~~

$Status_0$: ~~4J-4 to 4I-6~~
$Status_1$: 4J-3 to 4I-2, 2F-8, 20-2, 3J-4
$Status_2$: 4J-2 to 4K-18, 20-3, 3J-5
$Status_3$: 4J-1 to 4K-14, 2F-9, 20-1

$Depth_0$: 3K-5 to 4L-12, 20-10, 3L-2
$Depth_1$: 3K-6 to 4L-8, 20-7, 3L-4
$Depth_2$: 3K-13 to 4L-6, 20-8, 3L-7
$Depth_3$: 3K-14 to 4L-2, 20-1, 3L-1
$Depth_7$: 3K-15 to 2F-6

Bus Drivers

3E-17 to 4I-19, 4I-15, 4I-13, 4I-9
3E-12 to 4M-19, 4M-15, 4M-13, 4M-9
3E-9 to 4M-7, 4M-3, 4L-19, 4L-15
3E-8 to 1Q-19, 1Q-15, 1Q-13, 1Q-9
3E-5 to 2S-19, 2S-15, 2S-13, 2S-9
3D-20 to 2T-19, 2T-15, 2U-19, 2U-15
3D-17 to 2U-13, 2U-9, 2U-7, 2U-3
3D-12 to 1Q-7, 1Q-3, 2S-7, 2S-3
3D-9 to 1V-19, 1V-15, 1V-13
3D-8 to 1V-9, 1V-7, 1V-3
3D-5 to 3P-19, 3P-15, 3P-13
3C-20 to 3P-9, 3P-7, 3P-3
3C-17 to 4L-13, 4L-9, 4L-7, 4L-3
3C-12 to 4I-7, 4I-3, 4K-19, 4K-15
3C-9 to 4G-19, 4G-15, 4G-13, 4G-9
3C-8 to 4G-7, 4G-3
3C-5 to 4K-13, 4K-9, 4K-7, 4K-3
3B-8 to 3B-2
3B-1 to 3C-11, 3C-7

ALU

1G-3 to 1H-17
1G-4 to 1H-16
1G-5 to 1H-12
1G-6 to 1H-13
1G-7 to 1H-14
1G-8 to 1H-15

1H-18 to 2J-13, 2J-12
1H-19 to 2J-9, 2J-8
1H-20 to 2J-7, 2J-6
1H-1 to 2J-3, 2J-2

1J-3 to 1I-4
1J-4 to 1I-7
1J-5 to 1I-14
1J-6 to 1I-17
1I-5 to 1H-4
1I-6 to 1H-6
1I-15 to 1H-8
1I-16 to 1H-10

$OBus_0$: ~~2J-11 to 3K-9, 4J-6, 4N-6, 3M-6, 1S-7, 1T-7, 1U-7, 3O-6, 1R-19, 2O-20, 3S-7, 3T-7, 3U-7, 3V-20, 2N-12~~

$OBus_1$: ~~2J-10 to 3K-10, 4J-7, 4N-7, 3M-7, 1S-8, 1T-8, 1U-8, 3O-7, 1R-18, 2O-19, 3S-8, 3T-8, 3U-8, 3V-19, 2N-14~~

$OBus_2$: ~~2J-5 to 3K-17, 4J-8, 4N-8, 3M-8, 1S-9, 1T-9, 1U-9, 3O-8, 1R-17, 2O-18, 3S-9, 3T-9, 3U-9, 3V-18, 2N-16~~

$OBus_3$: ~~2J-4 to 3K-18, 4J-9, 4N-9, 3M-9, 1S-10, 1T-10, 1U-10, 3O-9, 1R-12, 2O-17, 3S-10, 3T-10, 3U-10, 2N-18~~

Scratch and Variable Memory

2R-1 to 2R-3, 2R-4, 2Q-10, 1R-6

mem$_0$: 1R-13 to 3Q-5, 1Q-18, 1P-3, 2O-14, 1L-17, 1I-9, 2Q-4
mem$_1$: 1R-14 to 3Q-6, 1Q-14, 1P-5, 2O-13, 2H-6, 1L-18, 1I-11, 2Q-7
mem$_2$: 1R-15 to 3Q-7, 1Q-12, 1P-6, 2O-12, 2H-5, 1L-19, 1I-7, 3N-4
mem$_3$: 1R-16 to 3Q-8, 1Q-8, 1P-8, 2O-11, 1L-20

Board Memory

22K resistor from 40-15 to 40-14
.01uf cap. from 40-13 to 40-12

3R-16 to 3Q-4, 4Q-10
3R-15 to 3Q-3, 4Q-11
3R-14 to 3Q-2, 4Q-12
3R-13 to 3Q-1, 4Q-13
2R-11 to 3R-6, 2Q-15, 2R-13, 2R-14
current_square: 4Q-14 to 4R-5, 2H-4, 1E-1

FA, LA

3B-9 to 3O-16
1E-13 to 2P-4
1E-8 to 2P-3
1E-3 to 2P-2, 2P-1
2P-5 to 2Q-2, 2Q-5, 3N-2

FA$_0$: 3O-4 to 3N-10, 3N-13, 3N-5, 2Q-3, 1E-2, 3P-18
FA$_1$: 3O-3 to 3N-11, 3N-15, 3N-6, 2Q-6, 1E-10, 3P-14
FA$_2$: 3O-2 to 3N-14, 3N-16, 3N-7, 3N-3, 1E-12, 3P-12

LA$_0$: 2Q-1 to 3P-8
LA$_1$: 2Q-8 to 3P-6
LA$_2$: 3N-1 to 3P-2

$\overline{\text{FAX}}$: 3N-9 to 2G-11
$\overline{\text{FAY}}$: 3N-12 to 2G-10
$\overline{\text{FAZ}}$: 3N-17 to 2G-9

FA positive: 3N-8 to 2F-4

MOVE, X, Y, Z

~~3G-15 to 3V-17~~
~~3V-5 to 2V-4~~
~~3V-4 to 2V-7~~
~~3V-3 to 2V-10~~
~~3V-6 to 2V-12~~
~~3V-2 to 2V-17~~
~~3V-7 to 2V-20~~
~~2V-3 to 3S-4~~
~~2V-6 to 3S-3~~
~~2V-11 to 3T-4~~
~~2V-14 to 3T-3~~
~~2V-15 to 3V-4~~
~~2V-18 to 3V-3~~
~~3A-9 to 3H-9~~
~~3A-8 to 3H-8~~
~~3A-5 to 3H-5~~
~~3H-11 to 3S-11~~
~~3H-6 to 3T-11~~
~~3H-3 to 3V-11~~
~~3G-11 to 4V-11~~

X_0: ~~3S-1 to 2S-18, 4Q-6~~
X_1: ~~3S-2 to 2S-14, 4Q-9~~
X_2: ~~3S-5 to 2S-12, 3R-1~~
X_3: ~~3S-6 to 2S-8, 3R-2~~

Y_0: ~~3T-1 to 2T-18, 3R-3~~
Y_1: ~~3T-2 to 2T-14, 3R-4~~
Y_2: ~~3T-5 to 2U-18, 3R-5~~
Y_3: ~~3T-6 to 2U-14, 3R-7~~

Z_0: ~~3U-1 to 2U-12, 3R-8~~
Z_1: ~~3U-2 to 2U-8, 3R-9~~
Z_2: ~~3U-5 to 2U-6, 3R-10~~
Z_3: ~~3U-6 to 2U-2, 3R-11~~

clear(x,y,z) depth: ~~4V-9 to 3S-12, 3T-12, 3U-12, 3K-12~~

Players Keys Selection

K_0: ~~4F-3 to 4G-18~~
K_1: ~~4F-4 to 4G-14~~
K_2: ~~4F-5 to 4G-12~~
K_3: ~~4F-6 to 4G-8~~
K_4: ~~4F-7 to 4G-6~~
K_5: ~~4F-8 to 4G-2~~

~~4F-20 to 4E-3~~
~~4F-18 to 4E-4~~
~~4F-16 to 4E-5~~
~~4F-14 to 4E-6~~
~~4F-12 to 4E-7~~
~~4F-10 to 4E-8~~

~~4F-19 to 4D-3~~
~~4F-17 to 4D-4~~
~~4F-15 to 4D-5~~
~~4F-13 to 4D-6~~
~~4F-11 to 4D-7~~
~~4F-9 to 4D-8~~

player 2:
K0: ~~4E-19 to 4C-2~~
K1: ~~4E-17 to 4C-1~~
K2: ~~4E-15 to 4B-8~~
K3: ~~4E-13 to 4B-7~~
K4: ~~4E-11 to 4B-6~~
K5: ~~4E-9 to 4B-5~~

player 1:
K0: ~~4D-20 to 4B-4~~
K1: ~~4D-18 to 4B-3~~
K2: ~~4D-16 to 4B-2~~
K3: ~~4D-14 to 4B-1~~
K4: ~~4D-12 to 4A-8~~
K5: ~~4D-10 to 4A-7~~

player 3:
K0: ~~4E-20 to 4C-8~~
K1: ~~4E-18 to 4C-7~~
K2: ~~4E-16 to 4C-6~~
K3: ~~4E-14 to 4C-5~~
K4: ~~4E-12 to 4C-4~~
K5: ~~4E-10 to 4C-3~~

player 0:
K0: ~~4D-19 to 4A-6~~
K1: ~~4D-17 to 4A-5~~
K2: ~~4D-15 to 4A-4~~
K3: ~~4D-13 to 4A-3~~
K4: ~~4D-11 to 4A-2~~
K5: ~~4D-9 to 4A-1~~

Ready to Mark fix

remove wire to 2C-14 from its chain
remove wire to 2C-13 " "
remove 2C-12 to 2C-19
remove 2C-16 to 2C-20

wire 2C-16 to 2C-15
wire $\overline{\text{clock}}$ to 2C-19 from 2J-17

Status bit 1 to random bit 1: remove wire to 2F-8
wire 4H-9 to 2F-8

---

mem bit 2 to AC2 bit 3: remove wire to 2H-5
wire 1J-20 to 2H-5

to replace two 600A's with 600 board

remove 600A boards 4A, 4B
insert 600 board in 4B

MOVE:
P2K0: 4E-19 from 4C-2 to 4B-18
P2K1: 4E-17 from 4C-1 to 4B-17
P2K2: 4E-15 from 4B-8 to 4B-16
P2K3: 4E-13 from 4B-7 to 4B-15
P2K4: 4E-11 from 4B-6 to 4B-14
P2K5: 4E-9 from 4B-5 to 4B-13
P1K0: 4D-20 from 4B-4 to 4B-12
P1K1: 4D-18 from 4B-3 to 4B-11
P1K2: 4D-16 from 4B-2 to 4B-10
P1K3: 4D-14 from 4B-1 to 4B-9
P1K4: 4D-12 from 4A-8 to 4B-8
P1K5: 4D-10 from 4A-7 to 4B-7
P0K0: 4D-19 from 4A-6 to 4B-6
P0K1: 4D-17 from 4A-5 to 4B-5
P0K2: 4D-15 from 4A-4 to 4B-4
P0K3: 4D-13 from 4A-3 to 4B-3
P0K4: 4D-11 from 4A-2 to 4B-2
P0K5: 4D-9 from 4A-1 to 4B-1

---

to wire up the mounted switches for player #3

P3K0	at	4C-8	to Normally open contacts
P3K1	at	4C-7	
P3K2	at	4C-6	Ground
P3K3	at	4C-5	
P3K4	at	4C-4	
P3K5	at	4C-3	

---

then plug teletype board into 4A

and wire data out 4A+16 to 4P-16

Random register changes:

remove logic 1 to 4H-15 (load random)

wire load from Random: 3E-17 to 3D-2
Load : 3D-1 to 4H-15
R4: 4H-19 to 4H-3
R5: 4H-18 to 4H-4
R6: 4H-17 to 4H-5
R7: 4H-16 to 4H-6
R0: 4H-10 to 4H-11
R1: 4H-9 to 4H-12
R2: 4H-8 to 4H-13
R3: 4H-7 to 4H-14

Paper tape reader changes

remove tape_channel_4: 4P-6 to 2E-9 (if there)

remove forwarddrive: 4P-13 to 40-11
remove busy_level: 4P-14 to 2E-10
remove tape_channel 5: 4P-7 to 3R-12
6: 4P-8 to 3R-17
7: 4P-9 to 3R-18
8: 4P-10 to 3R-19

UART LSI 1602 plugs into 4R crystal - 450.56 KHZ
wire clock to clock in 4P-19 to 4P-2
wire tape_channel_4: 4P-8 to 3N-18 data in at 4P-16
tape_channel_2: 4P-6 to 3N-19
tape_channel_3: 3N-20 to 2E-9
tape_channel_5: 4P-9 to 3R-19
tape_channel_6: 4P-10 to 3R-18
tape_channel_7: 4P-11 to 3R-17
tape_channel_8: 4P-12 to 3R-12
data_received: 4P-15 to 2E-10
received_reset: 4P-17 from 40-20
ground master_reset: 4P-18 and tristate enable 4P-13 to 4Q-8

to implement seperate clear depth and correct LA.

Disconnect 3G-9 from I_5
connect 3G-9 to I_4 (4K-12)
Disconnect 3K-12 from clear x y z
connect 1I-2 to I_{15} $\overline{I_{14}}$ (2D-5)
1I-1 to $\overline{I_5}$ (1A-19)
2C-13 to clock
3K-12 to clear depth (2C-12)
1I-3 to 2C-14

Disconnect 1I-12 from FA_2
1I-11 from Mem_1
1I-10 from FA_1
1I-9 " mem_0
1I-2 " FA_0
1I-1 " mem_2
1I-3 " 4S-5
1I-8 " 4S-6
1I-13 " 4S-7
4S-8 " 2Q-5
3N-18 " Tape channel 2
3N-19 " Tape channel 4
3N-20 " Tape channel 2 (2E-9)

Connect FA_1 to 3N-18
Mem_0 to 3N-19
mem_1 to 4S-7
FA_2 to 4S-6
3N-20 to 4S-5, 4S-4
mem_2 to 4S-3
FA_0 to 4S-2
4S-1 to 4V-18
4S-8 to 4V-19
4V-20 to 2Q-5
tape channel 2 to 4V-N7
" " 4 to 3A-2
3A-1 to 4V-N6
4V-N8 to 2E-9

Prom 3 L

Adress	before	After
EE	23	21
F0	21	23
FC	21	23

Prom 4 R

Address	before	After
00	A0	B0

write only

Software Description

The following is a description of what each routine is doing. The initialization routine loads a maze from paper tape, sets the statuses of all players, robots and bullets to inactive. The view generation routine is responsible for loading the shift register to generate lines and for computing the depth down a corridor each player and robot can see. The search routine searches for what each player can see and give the appropriate information to the display processor, search for what each robot can see and set flags indicating if he can see a player or bullet, and to destroy anyone who sees a bullet at a depth of zero. The update people routine scans the switches and moves players right, left, up and down, forward if possible, and fire bullets, the routine also moves robots forward and fires their bullets based on flags, and inactive robots and players are placed in a random spot in the maze when their forward switches are pressed. The update bullets routine is responsible for moving all active bullets forward and destroy all bullets that move forward into a wall. The robot routine oscillates exploding people and controls all robot functions. The robots fire bullets at any players they see and always try to move forward. The robots will turn in the following cases: It can turn into a corridor and a bullet is in the robots view, it can turn into a corridor and can not move forward, it can turn into a corridor and can go forward and randomly decides to go into the corridor, and when the robot can neither go forward or turn into a corridor the robot will turn in a random direction.

Next is a description of some of the basic algorithms used. This is done for the reader who wishes to understand the software in a vague way. To actually try to understand the microcode completely is not advised because there are many instances where the code is obscure. This is because of the limitation of 256 words to program.

The algorithm in the view generation routine takes advantage of the symettry of every corner. The routine finds the value of the eight squares surrounding the lower left corner to generate lines. Then the down axis is replaced by the left axis so that the next time through the loop the upper left corner lines are generated. This is done four times so that the down axis is returned to its prior value. The player is then moved forward one square to generate the next set of four corners. If after moving forward the square is filled in a flag is set to generate no more lines and the depth down the corridor is written to memory. After sixteen times of moving forward the player is back to his original location and the routine is done.

The search routine determines whether or not a player can see another object by checking that the two axis that are not the forward axis have identical locations in the maze and the difference along the forward axis is less than the depth. For example, if the person is facing in the x direction any object he can see must have identical y and z coordinates, furthermore the x coordinate of the object minus the x coordinate of the person must

be less than the depth the person can see down the corridor. This is done for every person and bullet. Once a player is determined to be seeing an object a series of tests are performed to determine if the object was a bullet and if it was at a depth of zero, robot or player. The display processor is passed the following information: who is seeing this object (stored in player #) the forward axis of the person (stored in FA) the down axis of the person (stored in memory) the forward axis of the object being seen (stored in AC1) the depth of the object being seen (stored in depth) the object being seen (stored in AC2 using the following code 0-3 player #, 4 - robot, 5 - bullet, 6 - exploding type 1, 7 - exploding type 2)

The main algorithm used in the robot routine is in choosing whether or not to turn. The algorithm chooses a random axis to turn, this axis must be neither zero or seven or equal to the forward axis or the inverse of the forward axis. This corresponds to a random selection between right, left, up and down. If the robot can turn in this direction and move into a corridor then it moves to the found section of code which decides whether or not the robot wants to move forward or turn in the direction just found. If the robot can't turn in this direction then another try for a good random turn is made. If after 64 tries at making a good turn it concludes that the robot can't turn in any direction in which case the robot will move forward or if it cant move forward will turn even though it won't be facing an open corridor.

Memory layout

The address into the scratchpad memory consists of 3 bits from player #, P2 and P3 Memory is of the same format for each player. The layout for a player is given below:

P2=0, P3=0 status: bit 3 - flag used in view generation
bit 2 - on if player is exploding
bit 1 - on if player is inactive
bit 0 - type of explosion

P3=1 X
P3=2 Y
P3=3 Z
P3=4 FA
P3=5 DA
P3=6 robot flags: bit 3: on if robot sees a bullet
bit 0: on if robot sees a player
P3=7 depth that player can see down corridor

P2=1-7, P3=0 bullet status: bit 1 on if bullet is inactive
P3=1 X
P3=2 Y
P3=3 Z
P3=4 FA

P2=1, P3=6 last value of other keys (forward, fire)
P3=7 last value of direction keys (left, right, up, down)

Software Listing

Initialization

Address	Statement	Microcode	
00	clear x,y,z, player #, P2, P3, depth	92	B0
01	next_char: get_ptr Goto 0E	80	03
02	again: if ¬ptr_ready goto again	E1	02
03	if tape channel 4 = 0 goto next_char	E2	01
04	write board memory	80	04
05	AC2 ← x	01	42
06	x ← AC2 plus 4	46	44
07	if ¬(oBUS = 0) goto next_char	E3	01
08	AC2 ← y	01	52
09	y ← AC2 plus 1	5D	25
0A	if ¬(oBUS = 0) goto next_char	E3	01
0B	AC2 ← z	01	62
0C	z ← AC2 plus 1	5D	26
0D	if ¬(oBUS = 0) goto next_char	E3	01
0E	new_positions: mem ← 2	00	23
0F	inc P2	81	00
10	if ¬(P2 = 0) goto new_positions	E5	0E
11	inc Player #, clear P2	82	40
12	if ¬(Player # = 8) goto new_positions	E6	0E

View generation

13	main_loop: if screen refresh = 0 goto bullets	F5	BF
14	reset screen, init, clear depth, P2, P3, Player #	92	A5
15	AC2 ← 8	00	82
16	wait_here: if display wait goto wait_here	C8	16
17	status ← mem, inc P3	05	3D
18	x ← mem, inc P3	05	34
19	y ← mem, inc P3	05	35
1A	z ← mem, inc P3	05	36
1B	FA ← mem, inc P3, shift	15	3A
1C	− ← mem, move	21	30
1D	LA → mem, move, shift	31	B3
1E	LA → −, move, shift	31	B0
1F	FA → −, move, shift	31	A0
20	LA → −, move, shift	33	B0

Address	Statement	Microcode	
21	$\overline{mem} \rightarrow -$, move, shift	33	30
22	LA $\rightarrow -$, move, shift, inc P3	35	B0
23	shift, inc P3	14	00
24	if player # > 4 goto skip6	C7	26
25	set display flag	80	02
26	skip6: if corner $\neq$ 3 goto skipover	C9	32
27	if current_square = 0 goto writeback	FF	2C
28	if status bit 3 = 1 goto writeback	CA	2C
29	depth $\rightarrow$ mem	01	C3
2A	clear P3	90	00
2B	mem $\leftarrow$ status or AC2	4F	D3
2C	writeback: P3 $\leftarrow$ 1	00	19
2D	X $\rightarrow$ mem	01	43
2E	inc P3	04	00
2F	Y $\rightarrow$ mem	01	53
30	inc P3	04	00
31	Z $\rightarrow$ mem	01	63
32	skipover: inc player #, clear P3	90	40
33	if player # $\neq$ 8 goto wait_here	E6	16
34	clear player #, inc depth	A0	80
35	if depth $\neq$ 0 goto wait here	ED	16
36	clear_loop: mem $\leftarrow$ mem $\cdot$ $\overline{AC2}$	77	33
37	inc Player #	80	40
38	if Player # $\neq$ 8 goto clear_loop	E6	36

Search

39	hold: if display wait goto hold	C8	39
3A	status $\leftarrow$ 0	00	0D
3B	loop: status $\rightarrow$ Player #, dec P3	09	D7
3C	mem $\rightarrow$ depth, inc P3	05	3C
3D	if mem $\neq$ 0 goto next_player	EE	82
3E	inc P3	04	00
3F	X $\leftarrow$ mem, inc P3	05	34
40	Y $\leftarrow$ mem, inc P3	05	35
41	Z $\leftarrow$ mem, inc P3	05	36

Address	Statement	Microcode	
42	FA ← mem, inc P3	05	3A
43	skip1: clear Player #, P2, P3	92	80
44	Dup_player: status → AC1	01	D1
45	AC1 ⊕ player #	65	70
46	if OBUS = 0 goto next_object	C3	7E
47	object_loop: if mem bit 1 = 1 goto next_object	DD	7E
48	AC2 ← 0	00	02
49	x → AC1, inc P3	05	41
4A	if mem = AC1 goto y	DC	4D
4B	if FA ≠ ±x goto next_object	D0	7E
4C	mem minus AC1 → AC2	59	32
4D	Y: Y → AC1, inc P3	05	51
4E	if mem < AC1 goto z	DC	51
4F	if FA ≠ ±y goto next_object	D1	7E
50	mem minus AC1 → AC2	59	32
51	Z: z → AC1, inc P3	05	61
52	if mem = AC1 goto match	DC	55
53	if FA ≠ ±z goto next_object	D2	7E
54	mem minus AC1 → AC2	59	32
55	match: if FA = + goto test	CF	57
56	AC2 ← 0 − AC2	5A	02
57	test: if AC2 > depth goto next_object	D3	7E
58	AC2 → depth, inc P3	05	2C
59	mem → AC1	01	31
5A	Play → AC2	01	72
5B	Player # → y	01	75
5C	P2 → z	01	86
5D	if OBUS = 0 goto player	C3	65
5E	clear P3, P2	92	00
5F	depth → −	01	C0
60	if OBUS ≠ 0 goto skip3	E3	63
61	player # ← status	01	D7
62	mem ← 5	00	53
63	skip3: AC2 ← 5	00	52
64	goto non_exploding	E0	6B
65	player: clear P2, P3	92	00

Address	Statement	Microcode	
66	if ¬(switches < player #) goto person	F7	68
67	AC2 ← 4	00	42
68	person: if mem = 0 goto non-exploding	CE	6B
69	AC2 ← 2	00	22
6A	AC2 ← AC2 plus mem	47	32
6B	non-exploding: status → player #	01	D7
6C	P3 ← 5	00	59
6D	if player # > 4 goto robot	C7	70
6E	set display flag	80	02
6F	holdup: if display wait goto holdup	C8	6F
70	robot: Y → depth, inc P3	05	5C
71	AC2 ⊕ 5 → AC1	66	57
72	if OBUS ≠ 0 goto skip4	E3	74
73	mem ← mem or AC2	4F	33
74	skip4: AC1 ← AC1 plus AC1	45	11
75	mem ← mem or AC1	4D	33
76	Y → AC2, inc P3	05	52
77	depth ← mem	01	3C
78	P3 ← 3	00	39
79	Z → AC1	01	61
7A	mem → Z, dec P3	09	36
7B	mem → Y	01	35
7C	AC1 → P2	01	18
7D	AC2 → Player #	01	27
7E	next_object: inc P2, clear P3	91	00
7F	if P2 ≠ 0 goto object_loop	E5	47
80	inc Player #, clear P2	82	40
81	if Player # ≠ 8 goto dup_player	E6	44
82	next_player: status ← status plus 1	5F	DD
83	if status bit 3 = 0 goto loop	EA	3B
84	clear P2, P3, player #	92	80

Update people

85	upd_peo_loop: status ← mem, inc P3	05	3D
86	X ← mem, inc P3	05	34

Address	Statement	Microcode	
87	Y ← mem, inc P3	05	35
88	Z ← mem, inc P3	05	36
89	FA ← mem, inc P3	05	3A
8A	inc P2	81	00
8B	AC2 ← other keys, inc P3	05	F2
8C	AC1 ← mem · $\overline{AC2}$	77	31
8D	mem ← AC2	01	23
8E	AC2 ← direction keys, inc P3	05	E2
8F	depth ← mem · $\overline{AC2}$	77	3C
90	mem ← AC2	01	23
91	clear P2, dec P3	8A	00
92	if AC1 bit 1 = 0 goto fire	F9	A4
93	if status bit 1 or 2 = 0 goto move	EC	AA
94	replace: x ← random	01	04
95	y ← random	01	05
96	z ← random	01	06
97	if current_square = 1 goto replace	DF	94
98	FA ← 4, dec P3	08	4A
99	mem ← 6	00	63
9A	fire_store: P3 ← 4	00	49
9B	FA → mem	01	A3
9C	P3 ← 3	00	39
9D	z → mem	01	63
9E	P3 ← 2	00	29
9F	y → mem	01	53
A0	P3 ← 1	00	19
A1	x → mem	01	43
A2	P3 ← 0	00	09
A3	mem ← 0	00	03
A4	fire: if AC1 bit 3 = 0 goto rotate	FB	AC
A5	if status bit 1 or 2 = 1 goto rotate	CC	AC
A6	AC1 ← 0	00	01
A7	find: inc P2, clear P3	81	00
A8	if P2 = 0 goto rotate	C5	AC
A9	if mem bit 1 = 0 goto find	FD	A7
AA	move: move, FA → -	21	A0

Address	Statement	Microcode	
AB	if current_square =0 goto fire_store	FF	9A
AC	rotate: depth → ACI	01	C1
AD	clear P2	00	08
AE	P3 ← 5	00	59
AF	if ACI bit 3 =0 goto left	FB	B1
B0	FA ← $\overline{LA}$	03	BA
B1	left: if ACI bit 2=0 goto up	FA	B3
B2	FA ← LA	01	BA
B3	up: if ACI bit 1 =0 goto down	F9	B7
B4	FA ⇔ depth	01	AC
B5	$\overline{mem}$ → FA	03	3A
B6	depth → mem	01	C3
B7	down: if ACI bit 0 =0 goto next_player	F8	F8
B8	FA → depth	01	AC
B9	mem → FA	01	3A
BA	$\overline{depth}$ → mem	03	C3
BB	next_player: P3 ← 4	00	49
BC	FA → mem	01	A3
BD	inc Player #, clear P3	90	40
BE	if Player # ≠ 8 goto upd_peo_loop	E6	85

Update bullets

BF	bullets: if bullet refresh =0 goto robots	F4	D3
C0	clear player #, P2, P3, reset bullets	92	86
C1	next_bullet: inc P2, clear P3	91	00
C2	if P2=0 goto next_play_bull	C5	D1
C3	mem → -, inc P3	05	30
C4	if oBUS ≠0 goto next_bullet	E3	C1
C5	x ← mem, inc P3	05	34
C6	Y ← mem, inc P3	05	35
C7	Z ← mem, inc P3	05	36
C8	FA ← mem, dec P3, move	29	3A
C9	if current square =0 goto live_bullet	FF	CB
CA	x ← 2, dec P3	08	24
CB	live_bullet: Z → mem	01	63

Address	Statement	Microcode
CC	dec P3	08 00
CD	Y → mem	01 53
CE	dec P3	08 00
CF	X → mem	01 43
D0	goto next_bullet	E0 C1
D1	next_play_bull: inc Player #, clear P2	82 40
D2	if player # ≠ 8 goto next_bullet	E6 C1

Robots

D3	robots: if robot refresh = 0 goto mainloop	F6 13
D4	clear Player #, P2, P3, depth, robot reset	92 A7
D5	rloop: ACl ← mem	01 31
D6	if ACl bit 2 ≠ 1 goto skip5	FA D8
D7	ACl ⊕ 1 → mem	64 13
D8	skip5: if ¬(Player # > switches) goto next_robot	F7 F6
D9	X ← mem, inc P3	04 00
DA	X ← mem, inc P3	05 34
DB	Y ← mem, inc P3	05 35
DC	Z ← mem, inc P3	05 36
DD	ACl ← mem, inc P3	05 31
DE	NOP	00 00
DF	skip6: inc P3	04 00
E0	mem → AC2	01 32
E1	0 → mem	00 03
E2	inc P2	81 00
E3	mem ← AC2 or 2	4E 23
E4	clear P2, dec P3, clear depth	8A 20
E5	start: FA ← random	01 0A
E6	if FA ≠ ±X goto good	D0 E9
E7	NOP	00 00
E8	if FA = ±Z goto start	F2 E5
E9	P3 ← 4	00 49
EA	FA ← FA ⊕ ACl	65 AA
EB	if FA ≠ ±X goto good2	D0 EE
EC	NOP	00 00

Address	Statement	Microcode	
ED	if $\overline{FA} = \pm z$ goto next	F2	F1
EE	good2: FA → mem, move	21	A3
EF	if current_square=0 goto found	FF	FC
F0	$\overline{FA}$ → −, move	23	A0
F1	next: inc depth	A0	00
F2	if depth≠0 goto start	ED	E5
F3	forward?: AC1 → −, move	21	10
F4	if current_square = 1 goto write	DF	F9
F5	undo: mem←AC1	01	13
F6	next_robot: inc Player #, clear P3	90	40
F7	if player # = 8 goto mainloop	C6	13
F8	goto rloop	E0	D5
F9	write: inc P3	04	00
FA	goto undo	E0	F5
FB	NOP	00	00
FC	found: $\overline{FA}$ → −, move	23	A0
FD	if AC2 bit 3 goto write	DE	F9
FE	if random bit 1 goto write	CB	F9
FF	goto forward?	E0	F3

Display Processor

My portion of the project was to design and build a device that would display the views of the first four players, the view generator not only had to draw the correct view of the maze, but also had to display any other objects that a player might see (other people, robots, bullets, or exploding people) This all had to be done in a time short enough so the displays wouldn't flicker.

The problem of generating four views at first seemed immense, but there are many symmetries that came to our aid. The actual drawing of the maze was made much simpler because it was such a regular object. Since there is no preferential axis the maze is rotationally symmetric. This means that if I can figure out how to draw one corner I have figured out how to draw the other three, since each is a 90° rotation of the one before it. Another very nice feature of the maze is that every corner, no matter where it is located in the maze, has only four lines max. attached to it. These line are exactly the same for each corner except for a rotation as mentioned above and a scale factor, since thing get smaller the farther away they are. Now to draw a view of the maze for a person I just have to find out which of those four line at my current corner should be drawn (get that information from the spart of the project) and then drawn those lines. Then I rotate my outputt by 90° and repeat the process. After I have done four corner I move 1 step (called a level) done the corridor and repeat the process. After finishing all the levels I am done with that persons view, see figure 1+2.

Since the process we go through is the same for each of the players it is possible to draw all 4 views at the same time. Now instead of deciding whether or not to draw a line on decides which scopes beams get turneed on. Since the view are generated concurrently enormous time is saved (takes ~ 8 ms to draw a view).

We have basicly figured out how to do half the problem, what is left is drawing the different things each person sees. It turns out that many of the same mechanisms used

in drawing the maze can be used in drawing people. What we would like to accomplish is to be able to make the object smaller as it gets farther away from up, just like the maze. Also it would be nice if people are place in the maze in such a way that it is possible to determine which way they are headed.

Since we did not have enough PROM space to change the size of people for every level, we decided that there would be 4 different sizes. A size change would occur when the $\log_2$ of the distance the person was away from you changes, ie 2, 4, 8. To solve the other problem was slighty more difficult. The reason was that there where 6 possible ways the person could be facing and only 4 walls to place him on. The solution we can up with is shown in figure 3. If he is facing left, right, up, down he is place on the correct wall pointing in the right direction. If he is facing you he is placed in the middle of the square pointing down. If he is facing away from you he is placed in the center of the square pointing up.

Now that we know what we want to do, the only problem left is to figure out how to do it. A block diagram of how I approached the problem is figure 4. The view generator consists of a PROM where all the important constants are stored (length of lines, corner locations + how to draw different figures). and a line drawer. The line drawer has two modes, a load position, and a draw line. In the load position mode, it takes the information on the bus (call it a) and can either put out the x,y pair (a,0.) (a,0) or (0,0) depending on the control lines. In the draw mode the information on the bus is taken as an offset from the current location. It then draws a line whose "length" is the value of the offset. The slope + direction of the line come from the slope pins and can be any of the following $x, -y, y, -x, xy, -xy, x-y, -x-y$.

Brief Dis. The basic way the project works is as follows. Initially we are reset by an init pulse. This resets the state reg and the state controller. The state reg for the address for the Prom. we load from the Prom the location of the first corner. Then we draw the first line whose length

is the output of the prom. (note for every state reg value there are two prom outputs, the current corner location + the length of the current line.) The slope of that line was determined by the last two bits of the state reg, which are called the line #. Then the corner location is reloaded into the line drawer and the offset for the new line is found. After drawing all four lines it again inc the state reg. This increments the corner number and sets the line number back to zero. The corner number controls the rotation generator – so the result is the same lines are drawn only rotated 90°. This corner drawing procedure repeats 4 times. Inc the state reg now, resets both the line # and the corner # and inc the Level number. This changes the corner location stored in the prom and the length of the lines. This level drawing continues for 16 times until the entire view of the maze is finished. At that point the machine changes to the people drawing mode.

All the information needed to draw a person is contain in the prom. They are stored as 4 bytes where each byte contains the length of the line and its slope. Since the people are drawn entirely by offsets an intial position for a person has to loaded in. This is determined by the Level he is at and his orientation code. The person is placed at (a,0) where a is the location of the corner of the level he is at. This is then rotated to put him pointed in the right direction. The figure is then drawn by four lines (lines can have 0 length). After all the people have been drawn the view generator is inited and begins to redrawn the view of the maze.

Detailed Dis - The rotation generator takes two 8 bit words, a (x,y) pair, and rotates it by 0°, 90°, 180°, or 270°. The amount of rotation is determined by a 2 bit rotation code which is always the corner number. Rotation is preformed by 2 eight bit 2→1 multiplexers and 16 selectable invertors. See schematic of display and rotation generator. If the intial position was (x,y) a 90° rot. is (y,-x), 180° (-x,-y), and 270 (-y,x).

Two things are important to notice, the first is that -x is formed by inverting each bit so that -0 ≠ 0. This means that the distance from -x to x is not 2x, but is 2x+1. This is important when figuring out the Lengths of lines. Finally the MSB of each DAC input is inverted making a change from 00 to FF (0 to -1) a change of 1, and making a change from 7F to 80 (127 to -127) a chang of 256

Line Generator - The line generator outputs the (x,y) pair to the rotation generator. (see schematic). It has an 8 bit input bus where position and offset information come from. To draw a line from a corner the location of the corner is put on the input bus and ~~either~~ load the two counters are either loaded with that value or cleared depending on where a (a,a) load a (a,0) load or a (0,0) load was preformed. Then the offset is placed on to bus and each counter is either inc, dec or left alone depending on the value of the slope. In addition an offset counter is also clocked. When it's value is equal to the value on the bus the process stops.

Buss Driver- This has two functions, to act as a buffer for the PROM, and to send the correct slope to the line drawer. The slope information can come from two places, it is either stored in the prom with the offsets (for people) or it is the value of the line number (lowest two bits of state reg) (for corners). When the slope information is stored in the Prom, in addition to placing those bits on the slope bus, the 3 MSB of the input bus (where the slope was stored) have to be planked low.

Prom - This is where are the need constants are stored. a "Map" of the prom is shown in figure 5.

State Reg - This is just basicly a loadable counter. In the maze mode it is used to determine where we are in generating a view. The lowest 2 bits are the current line we are one (hence called line number) the next two bits tell us which corner we are on

(corner number). The MS four bits are the level we are currently out. The corner number goes to the rotation generator and the level # / line # form the lower 6 bits of the prom address. For that reason they are to store the same type of information when the machine is in people mode.

Selector - Is used to load the state reg with the correct stuff in people mode. It first load the reg with the seen player level number and rotation code (gotten from the orientation generator). This is to get the correct intial position of the seen person. After that has been load by the line generator the state reg is reloaded by the selector, but this time with the number of the thing being seen and to log of the distance away - but the rotation remains the same. This enables you to fetch out of prom the offsets to draw the thing being seen.

Orientation - I am passed the seen person FA and the FA DA and LeftA of the seer. From that I determine the relative direction that object is facing and can therefor figure out the rotation code. (See schematic)

Line buffer - In the maze mode lines are sent to me a corner (4 lines) at a time. Since I draw the views for all 4 people at the same time I have to wait until I have all 4 corners (16 lines). What I do is latch the lines that are sent to me in a temp reg, each time a set becomes available. When the 4th corner is presented all 16 lines are latched into 2 8 bit shift Reg. The $Q_7 + Q_3$ output of each reg. contains the information on line 0. After that line is drawn both SR are shifted so that $Q_7 + Q_3$ contain whether line 1 is drawn etc. In addition this section has a line that it turns high if none of the views can see the current line (NOline).

Z generator - This is the section that controls the intensity of the four scopes. It's output depends on the state of the

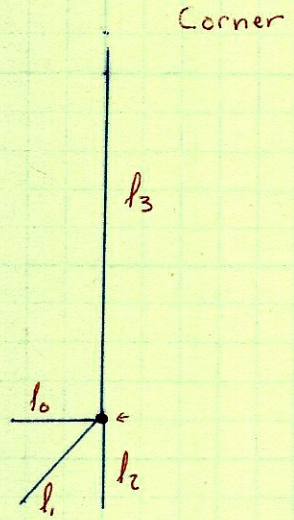
machine. In the maze mode the intensity is turned on if the line is present and that line is being drawn. In the people mode the beam is turn on only If it is currently drawing an object that you see.

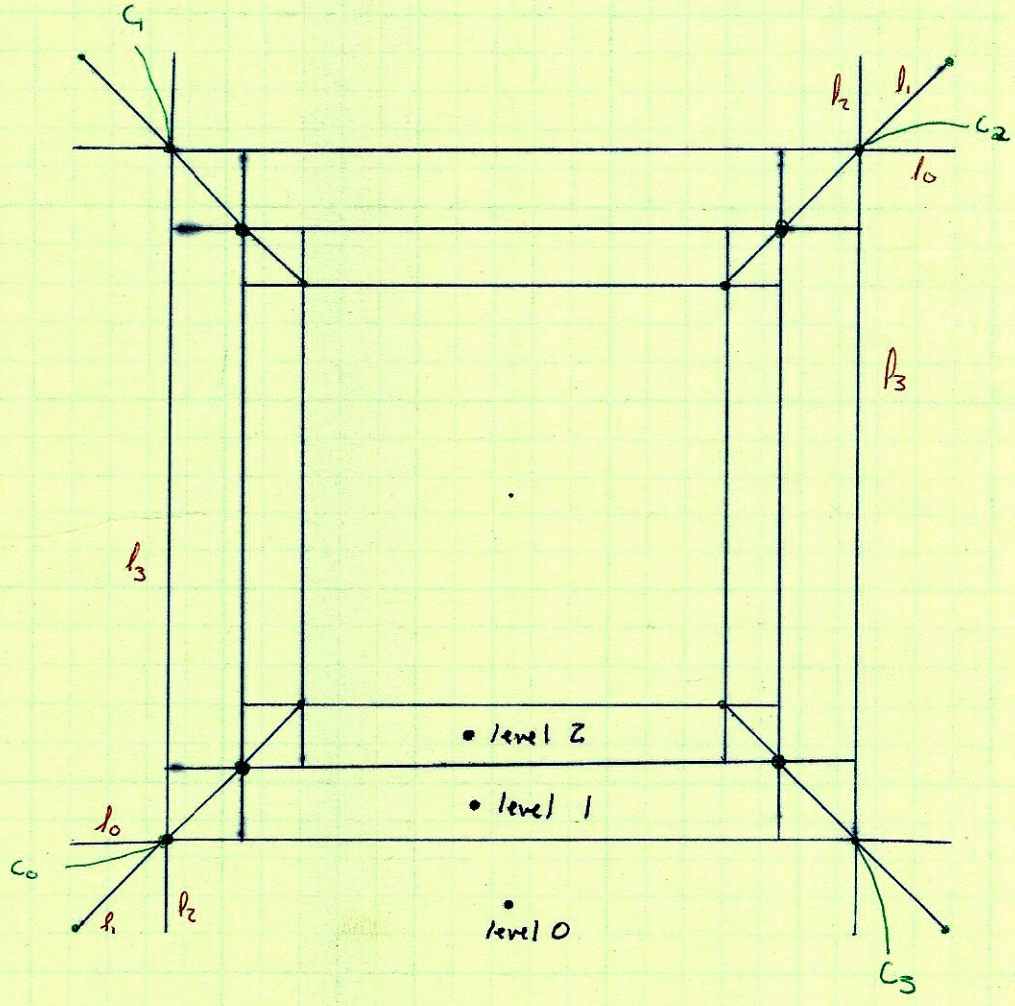
Bus Protocal- This is very simple, it is basicly a syn interface. When the other section of the project has data, they raise their ready line for one clock cycle. On the falling edge of that clock I raise my wait line. When my line is high the data must remain valid. When I have latched data the wait line is reset. In the passing of lines for the maze generation I will always be ready 3 of the 4 times, since the signals just go into temp reg. In that case I don't even raise my wait line

Control - See state diagram. The control sequence also has two major cycles, a maze section, and a people section. The basic sequence follow assuming we have just been initted. The intial state is T_0 and the state reg is reset. I waits in this state until DR goes high. This occurs when the first corner for all four have been recieved. The T_1 state loads the line generator with the current corner. T_2 is the actually state where the line is drawn. T_3 is used to decide which state you want to jump to, this depends on whether you have finish a corner or not. If you have then you have to wait for new information and hence go to T_0.

In the people mode the same process takes place except instead of jumping from T_3 to T_1 to reload the corner one jumps from T_3 to T_2 since you want to draw the new line from where the old line ended.

figure 1





Total Possible Lines for 3 Levels

figure 2

Possible View

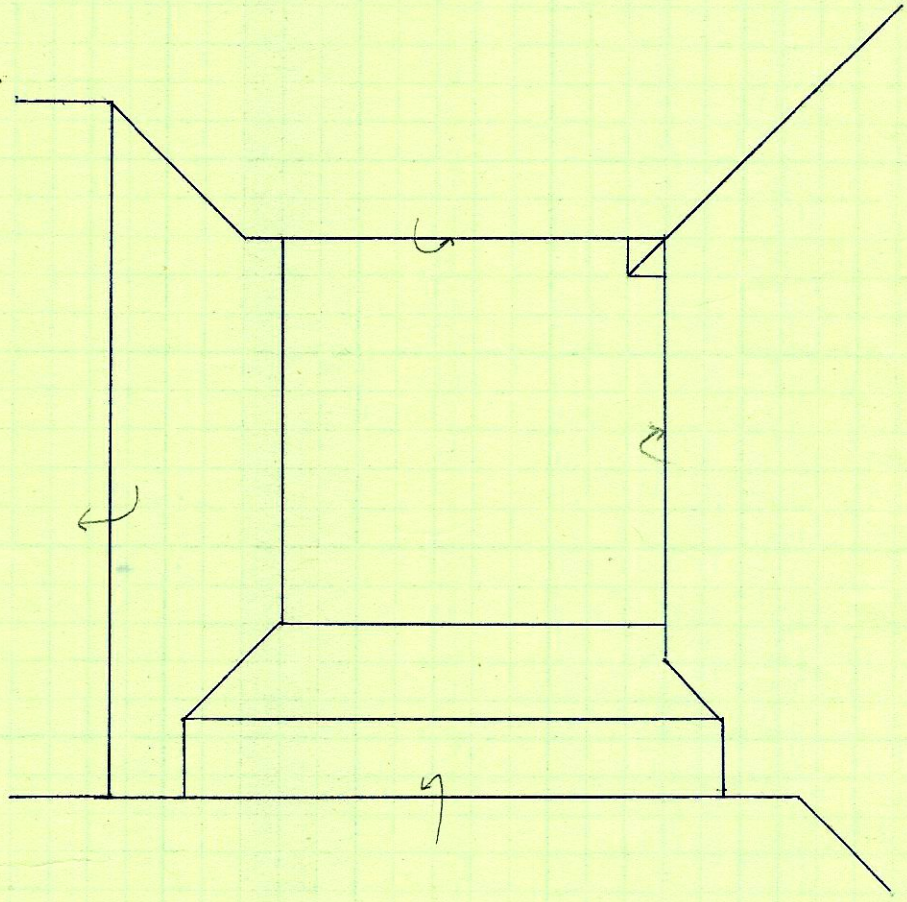
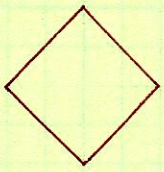


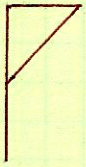
figure 3

What Things Look Like

Player 0

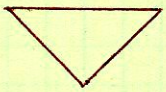


Player 1

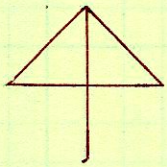


Player 2

Player 3

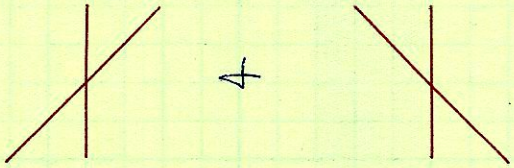


Robots 4

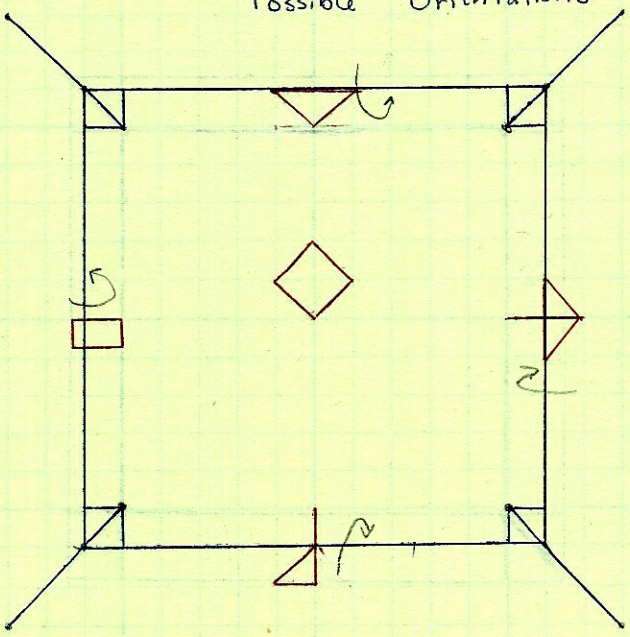


Bullets 5

Exploding People 7 6



Possible Orientations



Player	facing
0	~~look~~ away from you
1	down
2	left
3	up
Robot	right

Block Diagram of Display Generator

figure 4

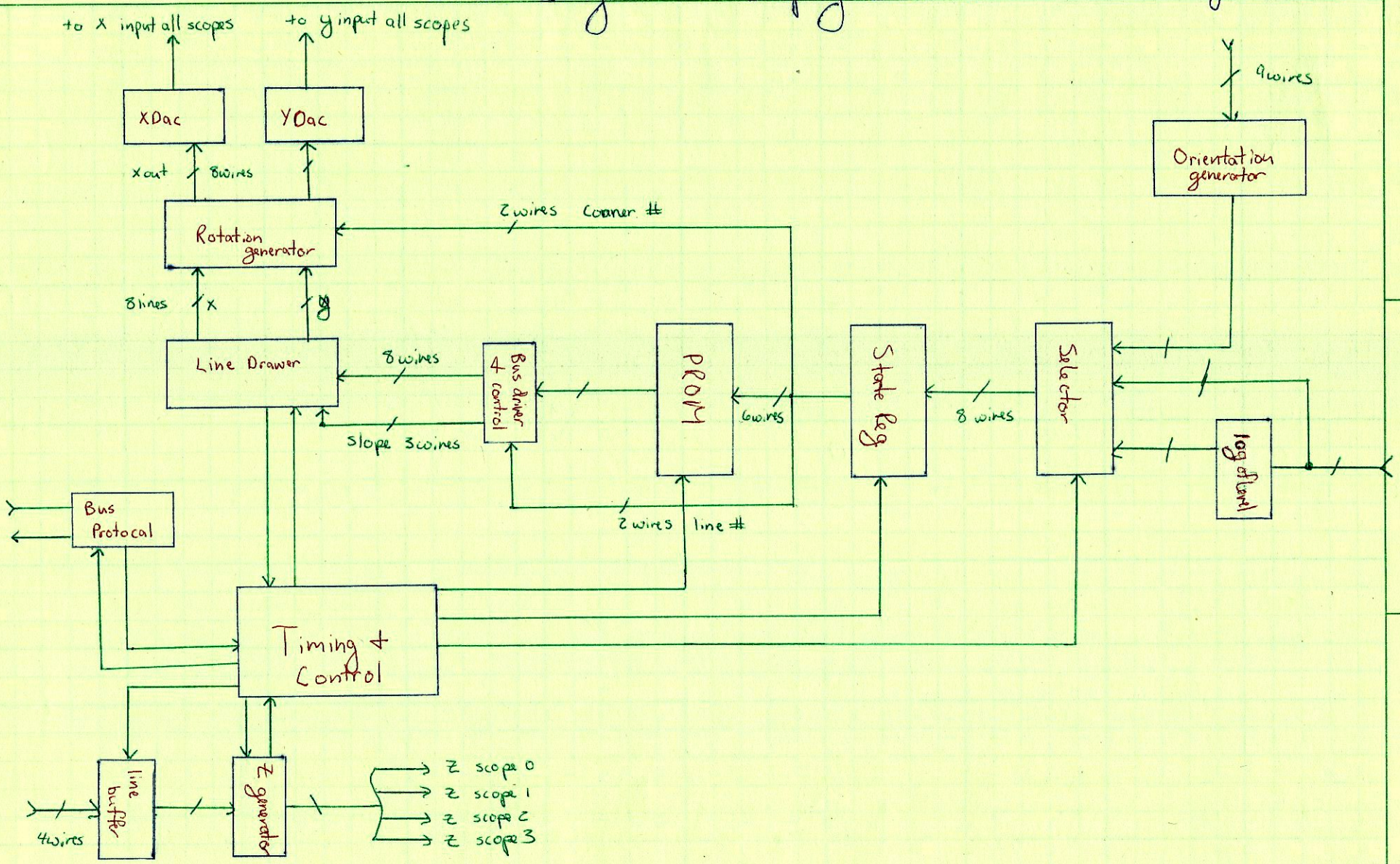
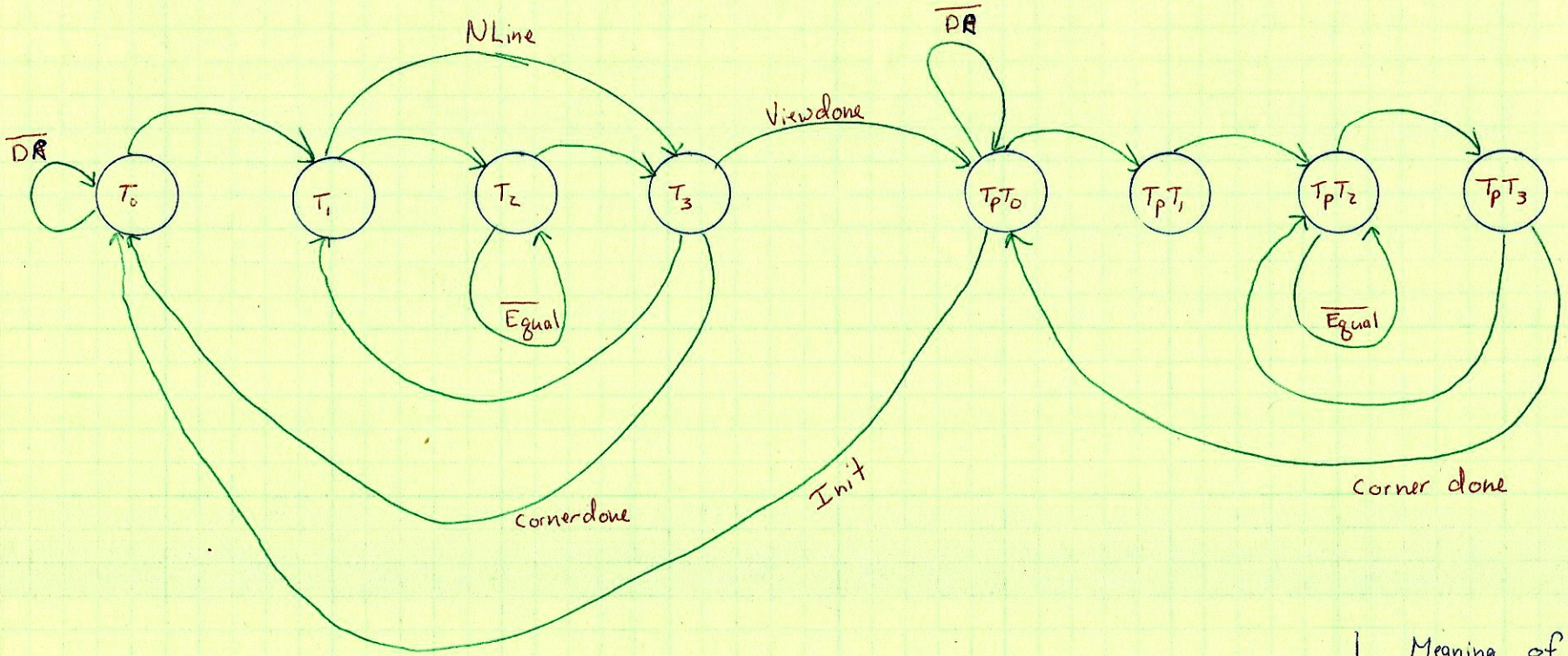


figure 5

PROM Map

Maze offset	length of lines for level 0	for level 1	level 2	level 3
	level 4	level 5	level 6	level 7
	8	9	A	B
	C	D	E	F
corner location	corner location for level 0	level 1	level 2	etc
			E	F
People robot	largest →			smallest
bullet	largest →			smallest
exp 1	"			"
exp 2	"			"
player 0	"			"
player 1	"			"
player 2	"			"
player 3	"			"

State Transition Diagram



	Binary form	Occurs during state
T_0	000	load X,Y reg
T_1	001	load X,Y reg
T_2	010	—
T_3	011	—
T_pT_0	100	load X,Y reg
T_pT_1	101	load X,Y reg
T_pT_2	110	—
T_pT_3	111	—

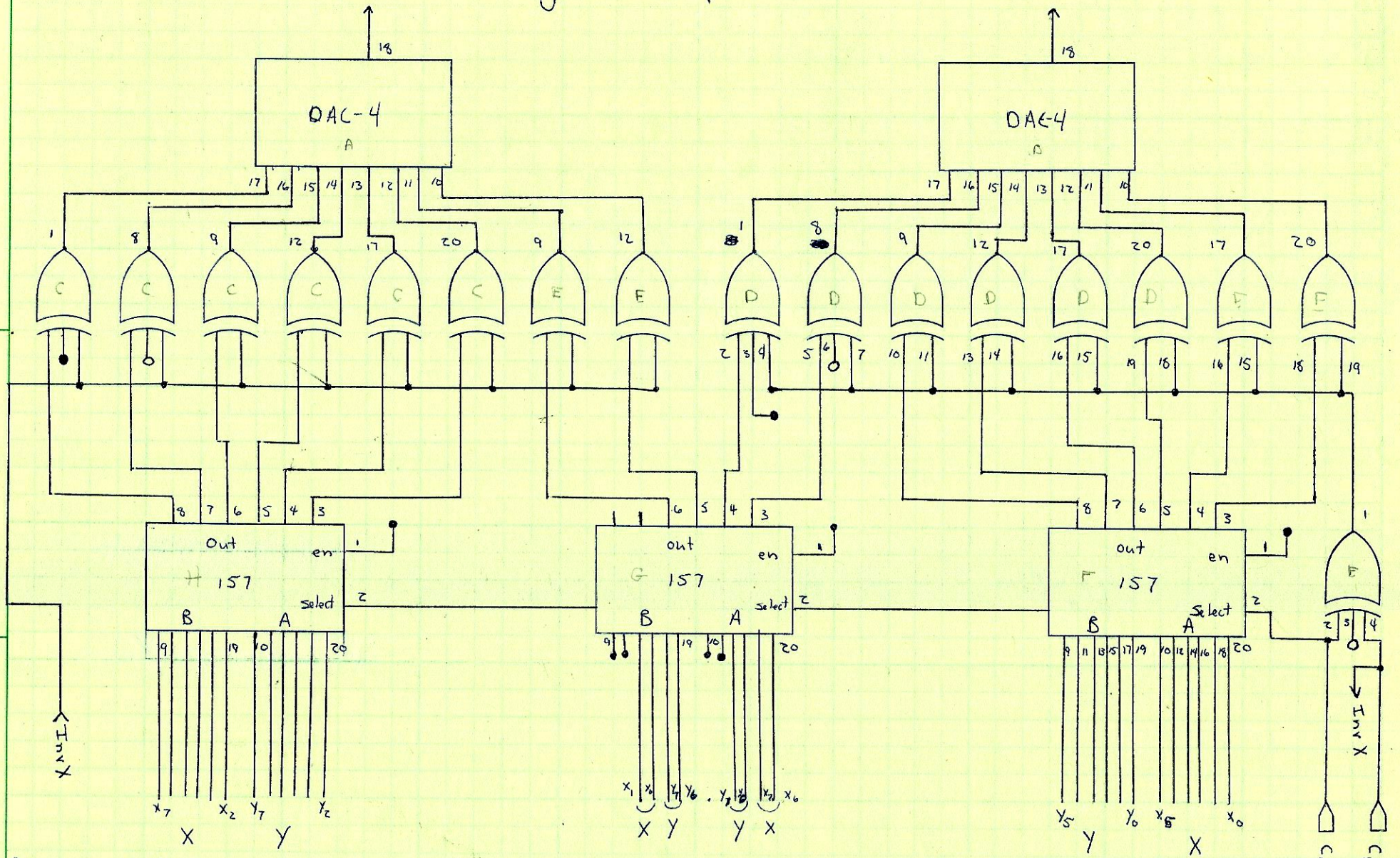
Occurs on transition

load lines from temp buffer
clear wait flag (occurs only when line# = 0)
inc offset clock X,Y Reg according to slope
inc state reg
load people information
reload state register, clear wait flag
inc offset, clock X,Y Reg according to slope
inc state reg

Meaning of lines

DA - is true when the needed information is received
Nline - true if all current lines are zero
Equal - true when offset counter is equal to the value of bus
Cornerdone - true when line# 3
Viewdone - true when state reg = FF

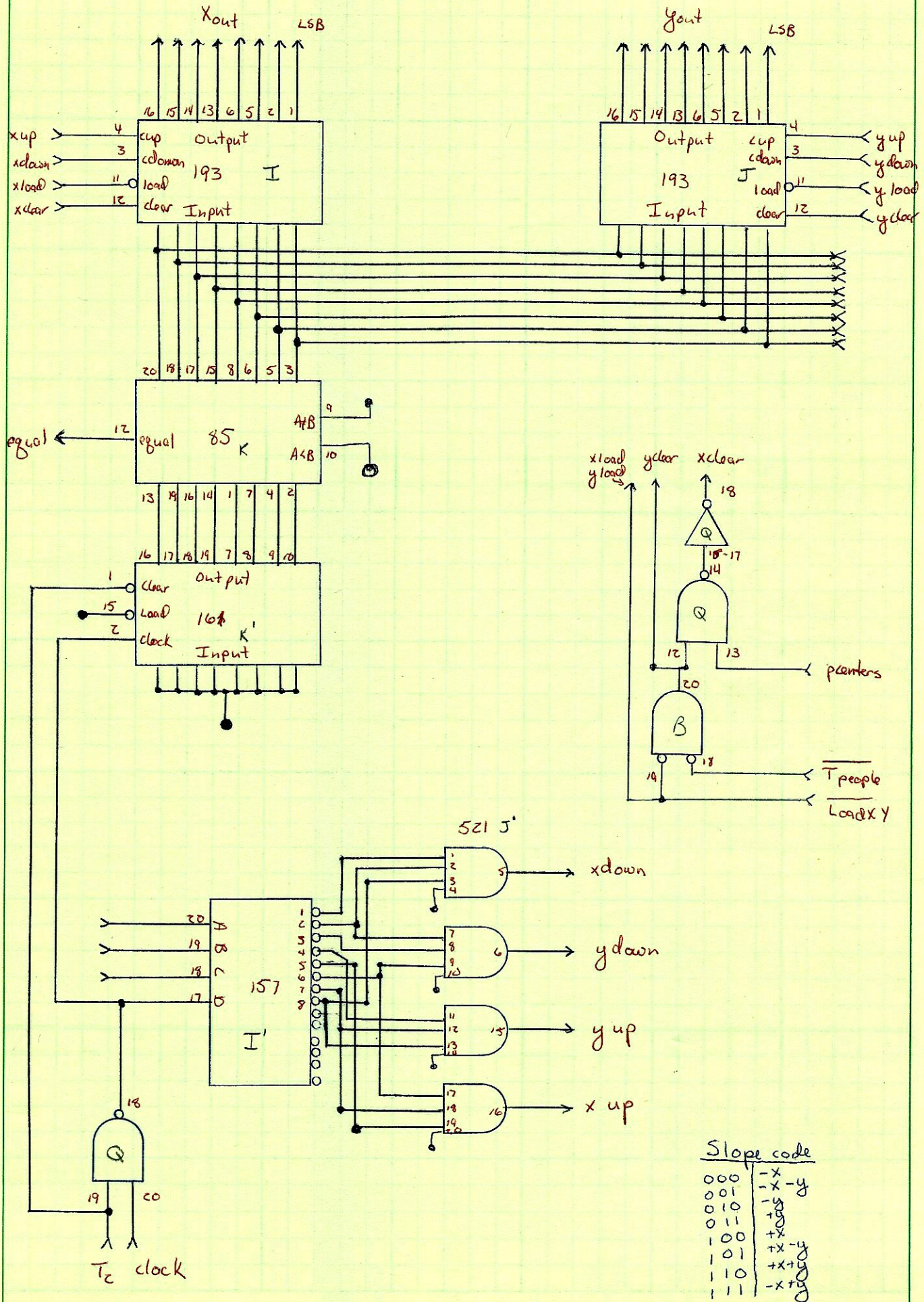
Display + Rotation Generator

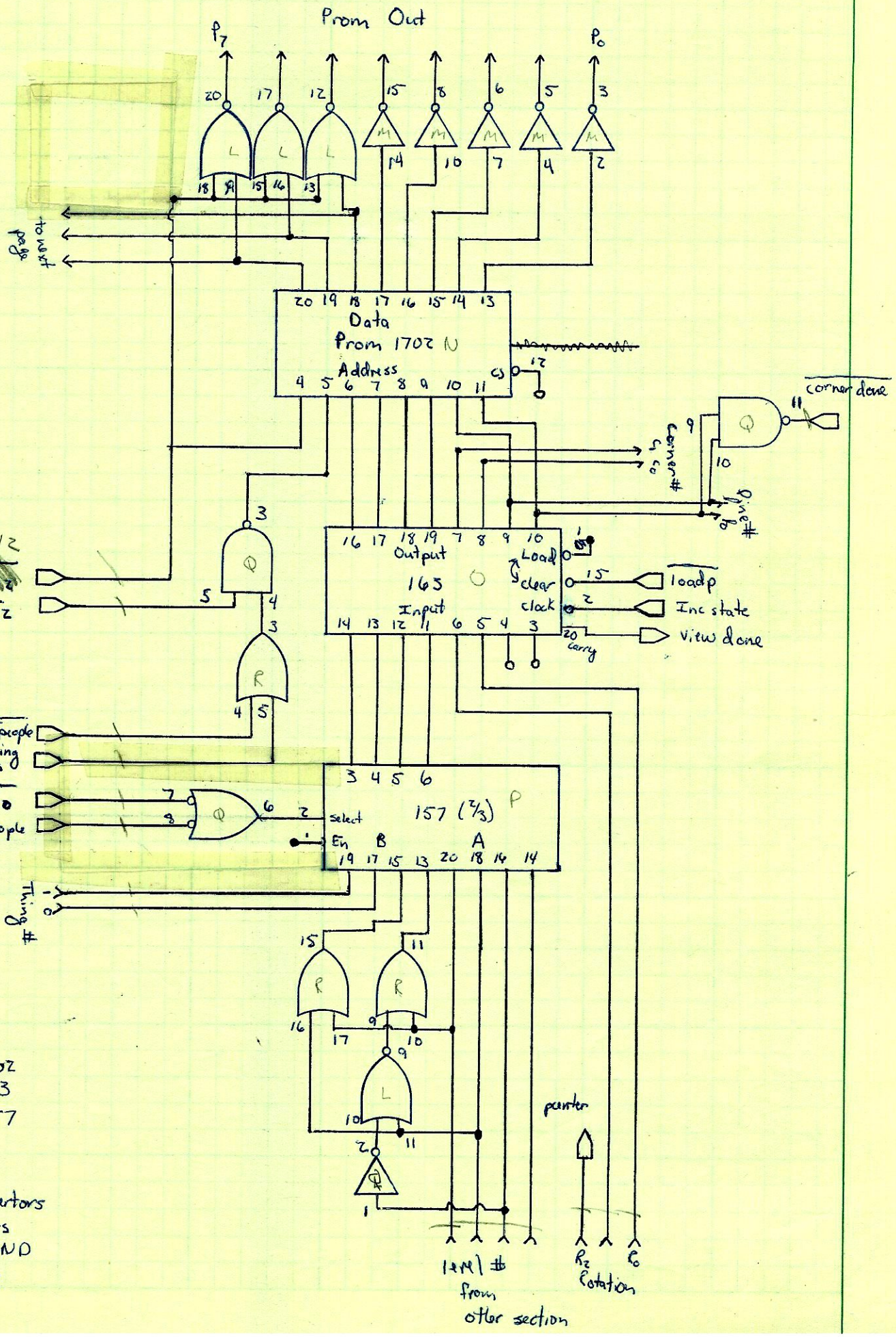


Parts 2 - Dac4
3 - MSI 86
3 MSI 157

from counters

Line Generator

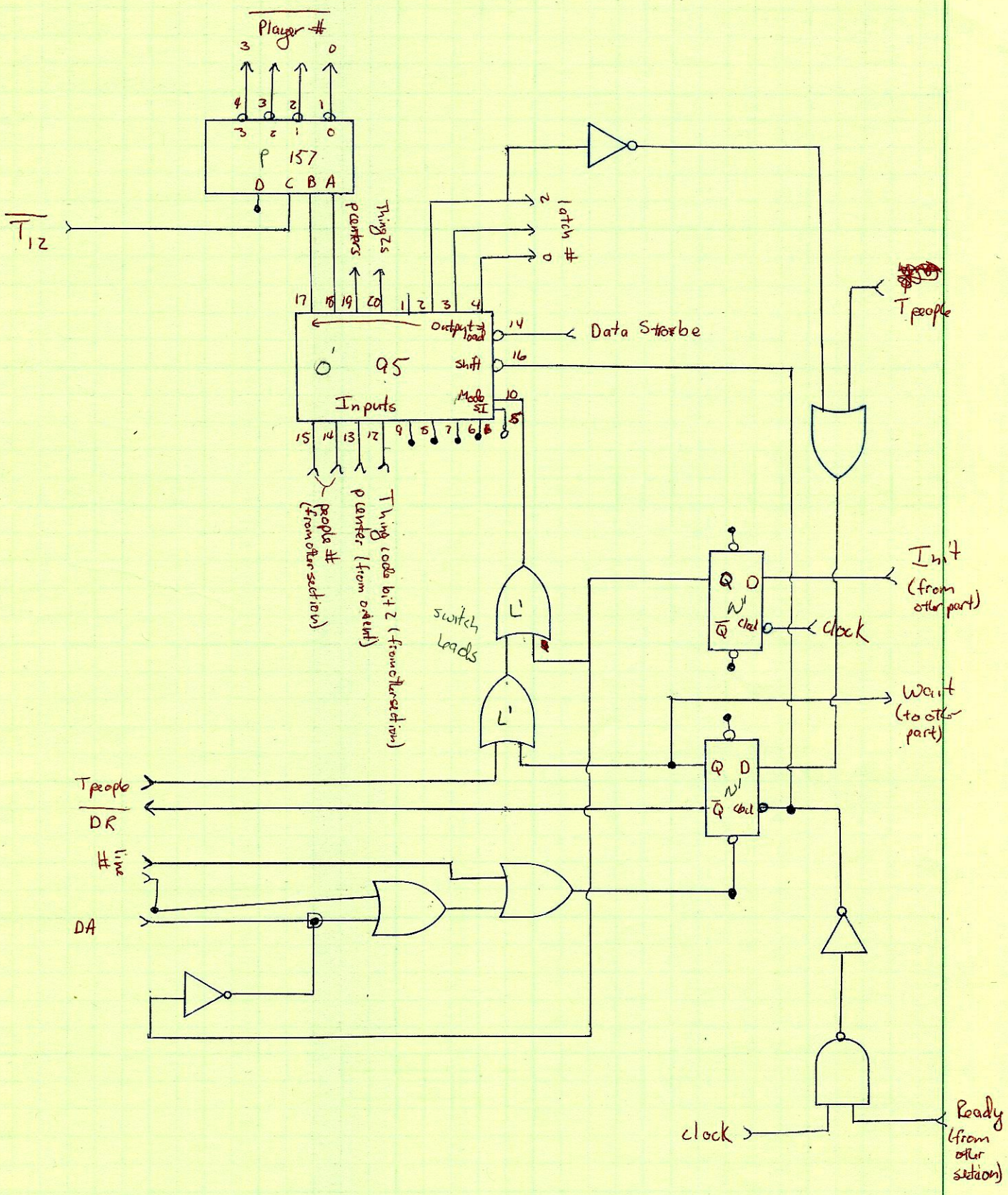




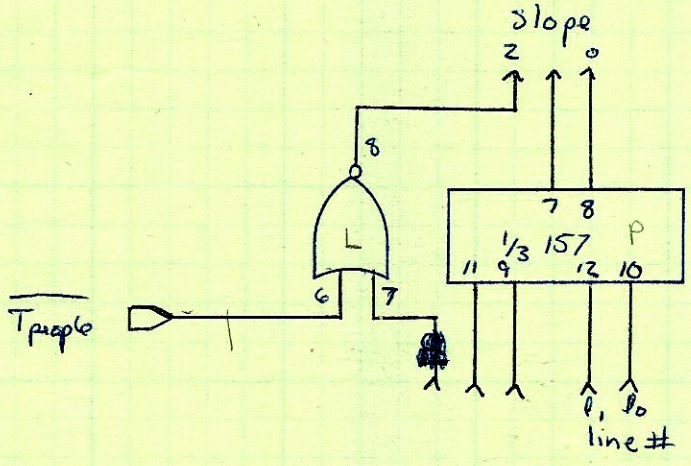
Parts
1 1702
1 163
1 157

Gates
6 Inverters
4 Nors
3 NAND
3 OR

Bus Logic + Control for Line Buffer



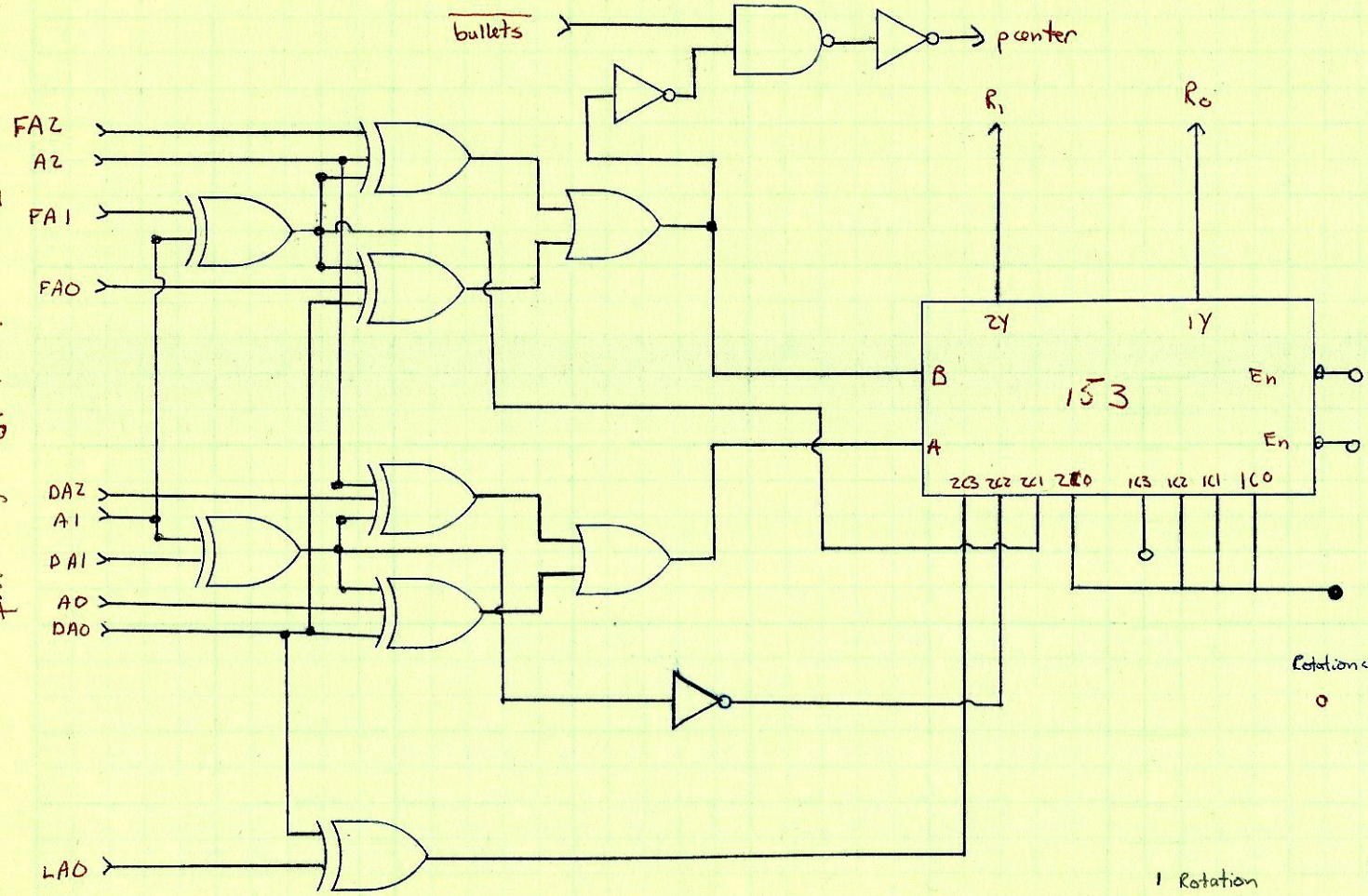
part of bus driver



Parts

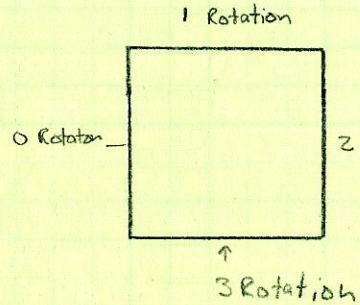
1 NOR

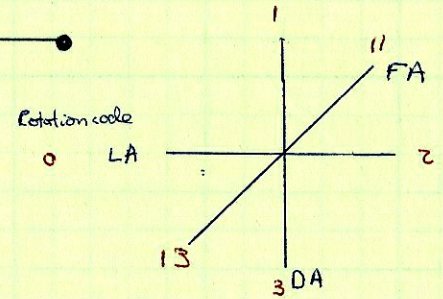
Orientation Encoder



From other half of project

FA → forward access of ~~sin~~ seeing player
DA → Down access of seeing player
LA → Left access of seeing player
A → forward access of player being seen.



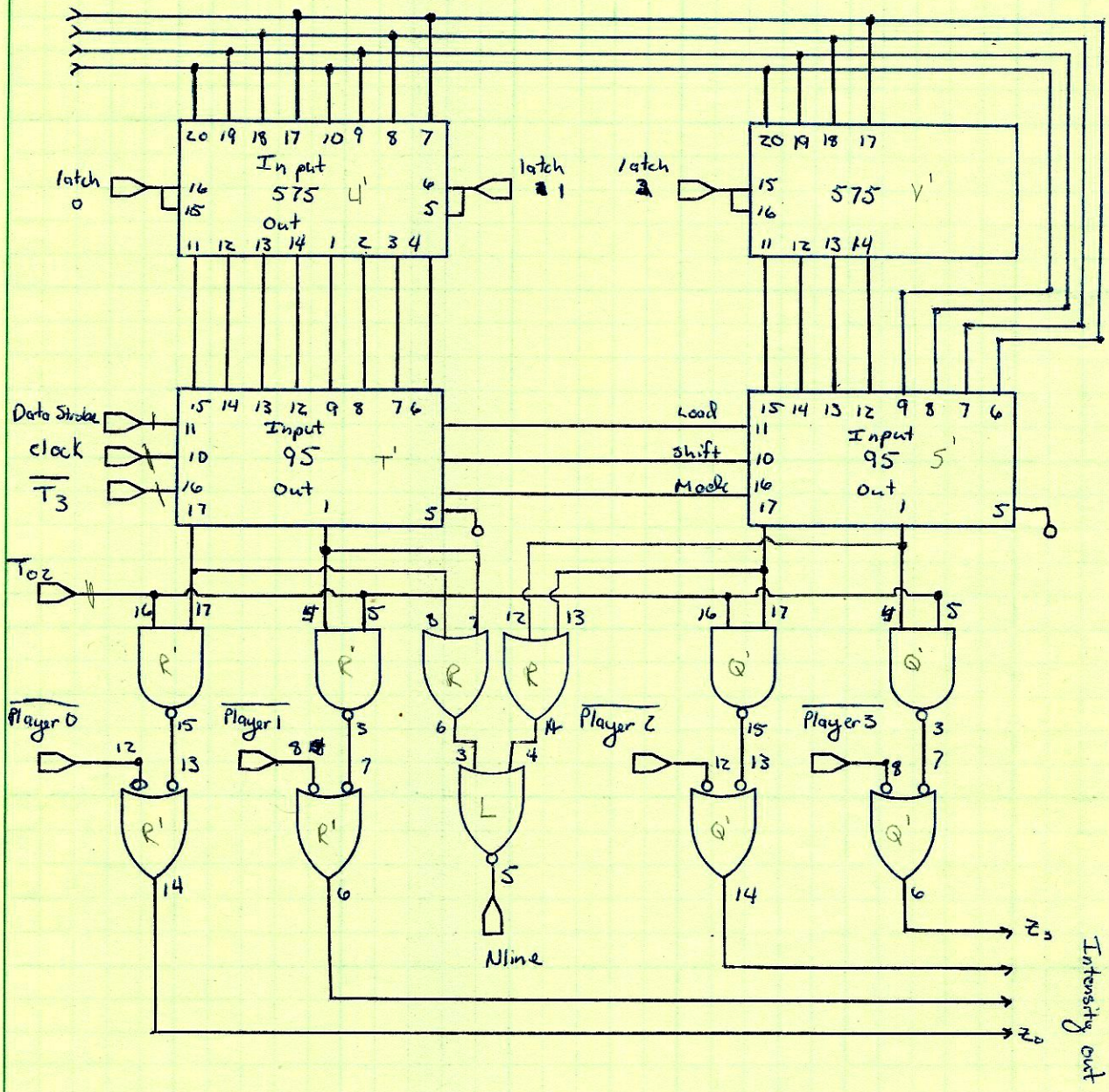


if the Forward access are in the same direction he should be ~~to~~ place in the ~~pla~~ center facing up or down R = 1 or 3

if the FA of the player see is equal (±) to the seeing players down access the rotation code is 1 or 3

if the FA is equal (±) to the seeing players left access the rotation code is 0 or 2

Scope Intensity Control + Line Buffer



Parts

2 575

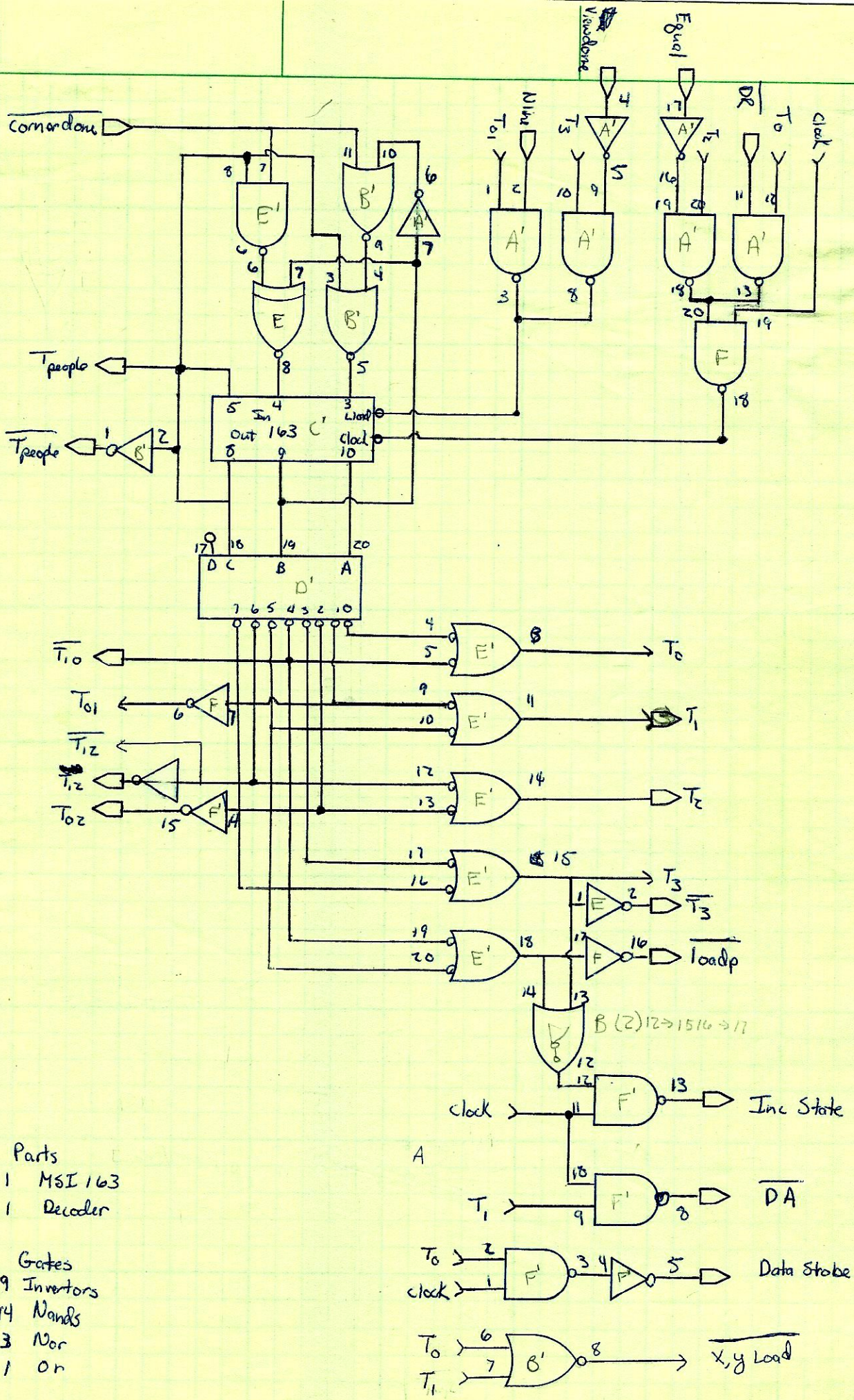
2 MSI 95

Logic

8 NANDs

2 OR

1 NOR



Control

Parts
1 MSI 163
1 Decoder

Gates
9 Inverters
14 Nands
3 Nor
1 Or

A

Wiring List

	Top		Bottom
1	A	Dac X	A 645
2	B	Dac Y	B 502
3	C	MSI 86 → X Dac	C MSI 163
4	D	MSI 86 Y Dac	D Recoder
5	E	MSI 86 Both	E 500 (1
6	F	MSI 157 → Y	F 645 (2
7	G	MSI 157 - Both	G Lights
8	#	MSI 157 - X	H Numbers display
9	I	MSI 193 X	~~I~~ Recoder
10	J	MSI 193 Y	~~J~~ 521
11	K	MSI 85	K MSI 161
12		~~MSI 161~~	L 532
13	L	502 ↖	M 600A
14	M	645 (Dor 6) +I	N 649 (FF)
15	N	1702	O MSI 95
16	O	~~MSI 163~~	P Recoder
17	P	MSI 157	Q 500 (2)
18	Q	500 ↖ (or 3)	R 500 (4) + Inve
19	R	532 (Nand 6) +I	S MSI 95
20	S		T MSI 95
21	T	MSI 186	U 575
22	U	MSI 86	V 575
	V	MSI 153	
		hack board + open collator	

Third group (W)

Wires

1	Init	11	level 3
2	Ready	12	line 3
3	Clock	13	Thing# 0
4	N Used	14	Thing# 1
5	level 0	15	Thing# 2
6	line 0	16	Rotation code 0
7	level 1	17	RC1
8	line 1	18	RC2
9	level 2	19	Who sees 0
10	line 2	20	Who sees 1

	01Y3 Corner Location		00Y00 Long vector		00Y (01 10 11) Short Vector	
0	71	91	E3	FC	0E	11
1	5A	BA	B5	AA	17	08
2	46	A6	8D	92	14	0B
3	38	D8	71	6E	0E	11
4	2E	CE	5D	42	0A	15
5	27	C7	4F	50	07	18
6	21	C1	43	5C	06	19
Y= 7	1D	FD	3B	24	04	1B
8	1A	FA	35	2A	03	1C
9	17	F7	2F	30	03	1C
A	15	F5	2B	34	02	1D
B	13	F3	27	38	02	1D
C	12	F2	25	38	01	1E
D	11	F1	23	3C	01	1E
E	10	F0	21	3E	01	1E
F	0F	EF	1F	00	01	1E

Real(--) Actual in Prom (inverted except last 3 bits)

Prom address 0 X Y Z

X = 1 ⇒ corner X = 0 ⇒ offset

Y = level # 0 → F

Z = line # 0 → 5

		Robot X=000				bullet X=001			
	0	A0	50	F0	90	B7	59	19	39
	1	AA	55	EA	95	BA	5B	1B	3B
Y=	2	B2	59	F2	99	BC	5D	1D	3D
	3	B7	5B	F7	9B	BD	5D	1D	3D

	X=100 People				X=101			
	90	10	30	50	A0	F0	50	FF
Y= 1	96	16	36	56	AA	F5	55	FF
2	99	19	39	59	B2	F9	59	FF
3	9B	1B	3B	5B	B7	FB	5B	FF

X=	110				X=	111			
	A0	F6	60	D6		94	E8	54	FF
	AA	F9	6A	09		98	F0	58	FF
	B2	FB	72	DB		9B	F6	5B	FF
	B7	FD	77	DD		9C	F8	5C	FF

X=	010				X=	011			
	A0	70	30	80		A0	70	10	40
	AA	75	35	8A		AA	75	15	4A
	B2	79	37	92		B2	79	19	52
	B7	7B	3B	97		B7	7B	1B	57

Prom address = 1XYZ

X = thing # 0 → 7

Y = depth 0 → 3

Z = line # 0 → 3

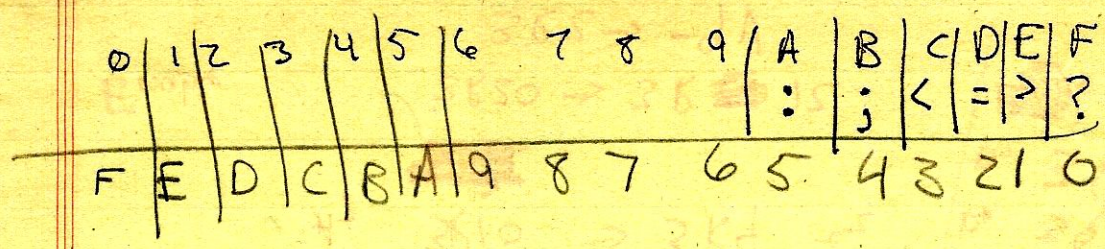
```
 D100 1FF
0100 FF FE 82 03 FE FE 82 A2 A2 BE A2 80 BF FF A2 82
0110 BE 82 82 82 82 82 FF FE 82 82 A0 A2 BF FF 02 20
0120 02 00 00 00 00 00 00 00 00 02 00 00 02 00 00 00
0130 02 00 00 00 00 00 00 00 02 00 00 20 00 00 00 20
0140 FE 00 83 FE B2 02 92 02 F2 06 82 02 FF 02 81 FF
0150 FF 02 81 02 81 02 81 02 83 02 82 22 FF FE 02 20
0160 02 00 00 00 00 00 00 00 00 04 00 00 02 00 00 00
0170 00 00 00 00 00 00 00 00 02 00 00 20 00 00 00 20
0180 FF BE 92 A2 97 A2 92 A2 97 E6 92 02 FF E2 82 22
0190 BF FE A2 02 AF 7E FA 02 8F 02 82 22 FF FE 00 20
01A0 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01B0 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01C0 03 FE 00 02 FF FA 80 0A BF EA A0 2A AF AA A8 AA
01D0 AF AA A8 2A AF EA A0 0A BF FA 80 02 FF FE 00 00
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00
01F0 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

*

For boxes

brown - ground
tan - right
grey - down
blue - up
green - forward
orange - left
yellow - fire

A - up
B - forward
C - down
D - left
E - fire
F - right



0 → Wall 1 space (so when it is inverted it turns out right.)

00 00 01 7D FC 01 01 7D 5D
08 5D 41 5D 7F 40 00 FD 7D
10 41 7D 7D 7D 7D 7D 00 01
18 7D 7D 5F 5D 40 00 FD DF

20 FD FF FF FF FF FF FF FF
28 FF FD FF FF FD FF FF FF
30 FD FF FF FF FF FF FF FF
38 FD FF FF FD FF FF FF DF

01 FF 7C 01 4D FD 6D FD
0D F9 7D FD 00 FD 7E 00
00 FD 7E FD 7E FD 7E FD
7C FD 7D FD 00 01 FD DF

60 FD FF FF FF FF FF FF FF
68 FF FB FF FF DF FF FF FF
70 FF FF FF FF FF FF FF FF
78 FD FF FF DF FF FF FF DF

D D D
00 41 6D 5D 68 5D 6D 5D
68 19 6D FD 00 1D 7D DD
40 01 5D FD 50 81 05 FD
70 FD 7D DD 00 01 FF DF

Changes for PROM Maze Memory

Remove

- ✓ 3R16 → 3Q-4 → 4Q10
- ✓ 3R15 → 3Q-3 → 4Q11
- ✓ 3R14 → 3Q2 → 4Q12
- ✓ 3R13 → 3Q1 → 4Q13
- ✓ 3R19 → 4P10
- ✓ 3R18 → 4P9
- ✓ 3R17 → 4P8
- ✓ 3R12 → 4P5

Memory Out Data

D_3 ↓ D_0

D_3 To memory ↓ D_0

✓ Write line 2R11 → 3R6 → 2Q5 →

only

Remove

If you want

4P-14 → 2E-10

4P-6 → 2E-9

Selector

Address ✓ X_0 4Q6 → 4Q3 | Enable (Z_3)

✓ X_1 4Q9 → 4Q2 | 3R11 → 4Q14

✓ X_2 3R1 → 4Q1 | Current square

[redacted] 4Q14 → 4Q1

Address lines

Change ✓ A_0 3R2 → 3R11 X_3

✓ 3R3 → 3R10 Y_0

✓ 3R4 → 3R9 Y_1

✓ 3R5 → 3R8 Y_2

✓ 3R7 → 3R7 Y_3

✓ 3R8 → 3R6 Z_0

✓ 3R9 [illegible]

✓ A_7 3R10 → 3R4 Z_2

4Q8 Remove

Output Put new wire

D_0 3R13 → 4Q11 ✓

D_1 3R14 → 4Q10 ✓

D_2 3R15 → 4Q9 ✓

D_3 3R16 → 4Q8 ✓

D_4 3R17 → 4Q7 ✓

[illegible] 4Q6 ✓

D_6 3R19 → 4Q5 ✓

D_7 3R20 → 4Q4 ✓

[redacted]

3R2 → -9V