CON

\_clkmode = xinput + pll8x

\_xinfreq = 8\_000\_000

COG2\_START = 0

VAR

OBJ

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

PUB start

coginit(0,@state\_cont,@@0)

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Memory map

'' ----------------------------------------------------------------------------------------------------------------------

'' $4000-$403f - Data from serial port. Bytewise.

'' $4040-$407f - Data to be written out to serial port. Bytewise, first byte is the length to

'' send. When non-zero the writer cog will send out data

'' $4080 - Mode control for receiver

'' $4084-$408f - State Controller for configuration decoding statemachine plus three words

'' of data used to exchange the cube config

'' $4090-$40DF - Used as scratch space for storing the lookup tables for the facelets

'' during encoding of the imaged cube.

'' $40E0-$40FF - Free

'' $4100 + 864l- Storage for pixel information to be displayed on the test screen

'' - $4E7F

'' $4E80 - $519B - Storage for camera colour data

'' $4E80 - $4E83 - stores which face is being read

'' $4E84 - $4EA7 - Stores average Y,V,U for Green Facelets

'' $4EA8 - $4ECB - Stores average Y,V,U for Blue Facelets

'' $4ECC - $4EEF - Stores average Y,V,U for White Facelets

'' $4EF0 - $4F13 - Stores average Y,V,U for Yellow Facelets

'' $4F14 - $4F37 - Stores average Y,V,U for Orange Facelets

'' $4F38 - $4F5B - Stores average Y,V,U for red Facelets

'' $4F5C - $4FBB - Stores Y,V,U for Green Centre Facelet

'' $4FBC - $501B - Stores Y,V,U for Blue Centre Facelet

'' $501C - $507B - Stores Y,V,U for White Centre Facelet

'' $507C - $50DB - Stores Y,V,U for Yellow Centre Facelet

'' $50DC - $513B - Stores Y,V,U for Orange Centre Facelet

'' $513C - $519B - Stores Y,V,U for Red Centre Facelet

'' $51A0 - $51B8 - Stores the average values for each face

'' $5D00-$5D5F - Storage for the starting cube configuration $5D20 - $5D5F

'' $5D00 - Flag for the virtual cube turner

'' $5D01 - $5D1F - Free

'' All at different times

'' $5D60-$5D7F - Stores info passed from permutation analyzer to SPI

'' $5D60 - SPI start command

'' $5D61 - COSET file to read from

'' $5D64-$5D78 - Human readables passed to SPI routines

'' $5D84-$5DCF - Stores the cube solution read from the COSET files

'' $5DD0-$5DFF - Free

'' $6000-$78C0 - Stores the data captured from the camera (88\*72 bytes)

'' ----------------------------------------------------------------------------------------------------------------------

'' Pin I/O

'' ----------------------------------------------------------------------------------------------------------------------

'' P40 Serial Data In Used during bootup and runtime

'' P39 Serial Data Out bootup and runtime

'' P38 Serial Data Comms to EEPROM

'' P37 Serial Clock Comms to EEPROM

'' P36 470 Ohm RCA OP

'' P35 270 Ohm RCA OP

'' P34 560 Ohm RCA OP

'' P33 1100 Ohm RCA OP

'' P32 VDD

'' P31 XO

'' P30 XI

'' P29 VSS

'' P28 HREF from camera

'' P27 VSYNC from camera

'' P26 Command to ARM2

'' P25 Command to ARM1

'' P24 Command to SERVO

'' P23 Digital Input to SD Card

'' P22 Clock Output to SD Card

'' P21 Digital Output to SD Card

'' P20 UV7 Data from camera

'' P19 UV6 Data from camera

'' P18 UV5 Data from camera

'' P17 UV4 Data from camera

'' P16 UV3 Data from camera

'' P15 UV2 Data from camera

'' P14 UV1 Data from camera

'' P13 PCLK from camera

'' P12 VDD

'' P11 RESET

'' P10 BOE

'' P9 VSS

'' P8 Y7 Data from camera

'' P7 Y6 Data from camera

'' P6 Y5 Data from camera

'' P5 Y4 Data from camera

'' P4 Y3 Data from camera

'' P3 Y2 Data from camera

'' P2 Y1 Data from camera

'' P1 Y0 Data from camera

'' ----------------------------------------------------------------------------------------------------------------------

'' COG MAP

'' ----------------------------------------------------------------------------------------------------------------------

'' COG 0 - Main Controller 0 (state\_cont) - this routine controls all the oter cogs

'' COG 1 - Video Ouput VID\_OP (vid\_op\_pro) - this generates NTSC signal from pixel

'' - Virtual Cube Turner VIRT\_CT (virt\_pro) - this routine uses moves pulled from lookup tables and keeps a virtual

'' version of the cube so the next set of moves can be calculated

'' COG 2 - Camera pixel data capture CAM\_CAP (cam\_cap\_pro) - this routine captures pixel data from camera

'' COG 3 - Camera colour data capture CLR\_CAP (clr\_cap\_pro) - this rotutine captures colour data from the camera

'' - Facelet Colour Classifier FC (fc\_pro) - this routine uses the the average colour data from the centre facelets

'' to make a best guess about the colour of each facelet

'' - External SD Card communications SPI (spi\_pro) - this routine reads data from the SPI card

'' this is used when retrieving lookup data for the

'' COSET mappings

''

'' COG 4 - RS232 Data Communication WRITER (writer\_pro) - this routine write RS232 communication data

'' to the PC

'' COG 5 - RS232 Data Communication READER (reader\_pro) - this routine reads data from the RS232

'' communication data from the host PC. Handles mode

'' change commands from the host PC

'' COG 6 - Actuator Driver ACTU (act\_pro) - this drivers the two linear and one servo actuator

'' COG 7 - Camera data to pixel data PRO4\_OP (pro\_4) - this routine converts the pixel data captures from

'' the camera and turns it into data to be used in

'' calculations

'' - Cube Solver STCB (cb\_st\_cnt) - this executes the cube solving algorithm

'' -----------------------------------------------------------------------------------------------------------------------

'' Scheduling

'' -------------------------------------------------------------------------------------------------------

'' Solution Steps

'' A) Command received (RST?) to start new solution - start and required cogs

'' B) Orient physical cube from starting position to scanning position

'' C) Scan faces - store a large sample of data for centre facelet and average for allother facelets

'' D) Determine cube configuration, i.e. determine facelts colours

'' E) Start Solution search

'' a) Look for COSET 0 solution using SPI look up tables

'' b) Once a partial soution is found, start actuators

'' F) Continue Solution search

'' a) Look for COSET 1 solution using SPI look up tables

'' b) Append partial soution

'' G) Continue Solution search

'' a) Look for COSET 2 solution using SPI look up tables

'' b) Append partial soution

'' H) Finish Solution search

'' a) Look for COSET 3 solution using SPI look up tables

'' b) Append partial soution

'' I) Idle waiting for restart

''

'' -------------------------------------------------------------------------------------------------------

'' APPROX STEP I A B C D D E E FGH

'' -------------------------------------------------------------------------------------------------------

'' VID\_OP |1|1|1|1| | | | | | | INTEGER REPRESENTS COG NUMBER

'' CAM\_CAP |2|2|2|2| | | | | | |

'' PRO\_4 |7|7|7|7| | | | | | |

'' CLR\_CAP | | |3|3| | | | | | |

'' VIRT\_CT | | | | |1|1|1|1|1|1|

'' FC | | | | |3|3| | | | |

'' STCB | | | | | | |7|7|7|7|

'' SPI | | | | | | |3|3|3|3|

'' ACTU | | |6|6| | | | |6|6|

'' WRITER |4|4|4|4|4|4|4|4|4|4|

'' READER |5|5|5|5|5|5|5|5|5|5|

'' CONTROLLER |0|0|0|0|0|0|0|0|0|0|

'' \ / \ / \ / \ / \ /

'' | | | | |

'' | | | | \--- START PHYSICALLY SOLVING CUBE

'' | | | \------- START SOLUTION CALCUALTION

'' | | \----------- FACE COLOUR DETERMINATION

'' | \--------------- COLOUR ANALYSIS

'' \------------------- IDLE

'' WHEN COMPLETE RETURN TO IDLE

''

''

'' RS 232 Data packet format is 0x01 CMD LENGTH DATA DATA .... 0x02

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' State controller.

'' Initialize DAT segments to run on particular cogs

'' Coordinates all other cogs

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

state\_cont '' Initialize parameter pointers

'' See introductory notes for the fucntion of each section of code

'' For each, take the current starting location, add it to the memory offset

'' stored in the labels at the end of the code location and Or in which COG

'' we need to run on

'' Purpose - Video Output (NTSC)

ADD VID\_OP,PAR

SHL VID\_OP,#2

OR VID\_OP,#$1

'' Purpose - Grab video data from the camera

ADD CAM\_CAP,PAR

SHL CAM\_CAP,#2

OR CAM\_CAP,#$2

'' Purpose - Extract from video snap shots, colour information related to the centre facelets

'' and average out all other facelt colours

ADD CLR\_CAP,PAR

SHL CLR\_CAP,#2

OR CLR\_CAP,#$3

'' Purpose - A virtual cube turner to allow the solution to continue without waiting for

'' the physical turner

ADD VIRT\_CT,PAR

SHL VIRT\_CT,#2

OR VIRT\_CT,#$1

'' Purpose - Based on average centre facelet measurements calculate a best guess for the colour

'' of each facelet

ADD FC,PAR

SHL FC,#2

OR FC,#$3

'' Purpose - Calcuates the cube configuration at each stage, encodes and reutrns what to look up

'' in the COSET

ADD STCB,PAR

SHL STCB,#2

OR STCB,#$7

'' Purpose - writes out data via serial port, only used for debugging

ADD WRITER,PAR

SHL WRITER,#2

OR WRITER,#$4

'' Purpose - reads data received from serial part, mainly for debugging but also used

'' to start the solving routine

ADD READER,PAR

SHL READER,#2

OR READER,#$5

'' Purpose - based on data from the COSET tables, runs the actuators to physically solve the cube

ADD ACTU,PAR

SHL ACTU,#2

OR ACTU,#$6

'' Purpose - to transfer video data gathered from the camera so it can be displayed on the video stream

ADD PRO4\_OP,PAR

SHL PRO4\_OP,#2

OR PRO4\_OP,#$7

'' Purpose - interfaces with the external SD Card

ADD SPI,PAR

SHL SPI,#2

OR SPI,#$3

'' Start of state machine to drive cube solver

''

'' State 0

'' -------

'' Initialize -

'' a) Start the camera capture, data transfer and video output

'' b) Start the RS232 reader and writer routines

'' Otherwise just wait for command to start

''

'' State 1

'' -------

'' Scan cube for permutation i.e. the colour of the facelets

'' a) Orient the cube to predefined position

'' b) Go through faces in succession, and store data fro the centres

'' c) Calculate the average colour componets for the remaining facelets

'' d) Determine what the best guess for each facelet must be based on centre colours

'' e) Re-orient cube to predefined position before solving

'' f) Start COGS to

'' i) Calcualte cube permutations and encode them to pre-defined formats

'' ii) Start the SPI based look up from tabels stored on the SD card

'' iii) Start the virtual cube turner which will be used to predict the cube

'' state after cube solver provides next moves

''

'' State 2,3,4

'' -----------

'' These states do nothing at present. did some experiments with Neural Network classifiers

'' that would have been run in these states. For the interested reader - contact the author

'' to see primitve versions of this code

'' State 5

'' -------

'' Move the cube from G0 to G1

'' a) Calculate G0 permutation code

'' b) If this the cube is 'solved' in terms of G0, skip to the next state

'' c) Look for the permutation, and moves to G1, in the SD card lookup tables

'' d) Once found mechncial solver will start automagically executing the moves

'' e) Start the virtual turner to see the result of the moves found in the lookup table

''

'' State 6

'' -------

'' Move the cube from G1 to G2

'' a) Calculate G1 permutation code

'' b) If this the cube is 'solved' in terms of G1, skip to the next state

'' c) Look for the permutation, and moves to G2, in the SD card lookup tables

'' d) Once found mechncial solver will start automagically executing the moves

'' e) Start the virtual turner to see the result of the moves found in the lookup table

''

'' State 7

'' -------

'' Move the cube from G2 to G3

'' a) Calculate G2 permutation code

'' b) If this the cube is 'solved' in terms of G2, skip to the next state

'' c) Look for the permutation, and moves to G3, in the SD card lookup tables

'' d) Once found mechncial solver will start automagically executing the moves

'' e) Start the virtual turner to see the result of the moves found in the lookup table

''

'' State 8

'' -------

'' Move the cube from G3 to G4 (solved)

'' a) Calculate G3 permutation code

'' b) If this the cube is 'solved' in terms of G3, skip to the next state

'' c) Look for the permutation, and moves to G4, in the SD card lookup tables

'' d) Once found mechncial solver will start automagically executing the moves

'' e) Start the virtual turner to see the result of the moves found in the lookup table

''

'' State 9

'' -------

'' Idle Mode

'' State modes are cycles through sequentially, the variable ST holds the state

'' State 0 - Initialize

state\_0 SUB ST,#0 nr,wz

if\_nz JMP #state\_1

' Start serial data sender - not really used except for debugging

COGINIT WRITER

' Start serial data reader - starts the solver based on PC command but

' otherwise just used for debugging

COGINIT READER

' Start ntsc driver toput images on TV

COGINIT VID\_OP

' Start captue of image from camera

COGINIT CAM\_CAP

' Start transfer of luminosity data to pixel data

COGINIT PRO4\_OP

' Wait for a command to start

state\_0a RDBYTE ST\_0,MD\_CTRL wz

if\_z JMP #state\_0b

MOV ST,#1

JMP #state\_0

state\_0b MOV ST\_0,#1 ' While idling keep camera

WRBYTE ST\_0,CAP\_RT ' running, camera normally operates

JMP #state\_0a ' in a 'snapshot mode'

'' State 1 - Scan the cube to sample colours

state\_1 SUB ST,#1 nr,wz

if\_nz JMP #state\_2

' The the solver that physically solves the cube

' for this state this COG will be given single instructions to rotate

' the cube to the correct position so it's colours can be scanned

COGINIT ACTU

' Start captue of colour data from camera, this routine, captures

' pixels in the centre facelt and averages for all the other facelets

COGINIT CLR\_CAP

' Move to scanning configuration

' I could get rid of this by having the scanner store data in

' a different order

MOV ST\_0,#$24 ' This encodes the face to have in the cup

CALL #nxt\_face ' this executes a signle move command

MOV ST\_0,#$25

CALL #nxt\_face

MOV ST\_0,#$26

CALL #nxt\_face

' Get each face F,L,B,R,U,D to the top so the facelets can be

' scanned. The actuator controller takes commands so as to orient

' a particualr face into the cup. For this portion of the algorithm

' we need to the face in question to be pointing towards the camera

' so we send the opposite face command, putting that face in the cup

' and the face we want pointing to the camera.

MOV ST\_6,#0

MOV ST\_0,#3 ' Store the 'White' face

CALL #get\_clrs

'' Flip to the right face down

MOV ST\_0,#$21

CALL #nxt\_face

MOV ST\_0,#2 ' Store the 'Blue' face

CALL #get\_clrs

'' Flip to the front face down

MOV ST\_0,#$25

CALL #nxt\_face

MOV ST\_0,#4 ' Store the 'Yellow' face

CALL #get\_clrs

'' Flip to the left face

MOV ST\_0,#$22

CALL #nxt\_face

MOV ST\_0,#1 ' Store the 'Green' face

CALL #get\_clrs

'' Flip to the down face

MOV ST\_0,#$24

CALL #nxt\_face

MOV ST\_0,#5 ' Store the 'Red' face

CALL #get\_clrs

'' Flip to the up face

MOV ST\_0,#$23

CALL #nxt\_face

MOV ST\_0,#6 ' Store the 'Orange' face

CALL #get\_clrs

'' Use the facelet classifier to guess which facelts are which colours

COGINIT FC

MOV ST\_0,#1

WRBYTE ST\_0,FC\_GO

'' Flip back to the starting position - yellow face in the cup, blue face

'' pointing towrd actuator 2 (fixed) and green pointing toward actuator 1.

MOV ST\_0,#$22

CALL #nxt\_face

MOV ST\_0,#$23

CALL #nxt\_face

MOV ST\_0,#$21

CALL #nxt\_face

MOV ST\_0,#$26

CALL #nxt\_face

' Start cog to calcualte cube configurations - this takes information

' about the cube permutation and encodes it into the format stored in

' the lookup tables on the SD card

COGINIT STCB

' Initialize the SPI card - the COSET lookup tables are all stored

' on the SD card. communication to the card is using SPI protocol

COGINIT SPI

' Start the virtual cube turner - this routine allows the cube solution

' to proceed without waiting for the physical solution and then rescanning

COGINIT VIRT\_CT

st1\_next MOV ST,#2

JMP #state\_0

'' State 2 - Spare state

state\_2 SUB ST,#2 nr,wz

if\_nz JMP #state\_3

st2\_next MOV ST,#3

JMP #state\_0

'' State 3 - Spare state

state\_3 SUB ST,#3 nr,wz

if\_nz JMP #state\_4

st3\_next MOV ST,#4

JMP #state\_0

'' State 4 - Spare state

state\_4 SUB ST,#4 nr,wz

if\_nz JMP #state\_5

st4\_next MOV ST,#5

JMP #state\_0

'' State 5 - Move the cube from G0 to G1

state\_5 SUB ST,#5 nr,wz

if\_nz JMP #state\_6

'MOV ST\_3,CB\_CONFIG

'CALL #st\_s

' Start the state machine that encodes the cube permutations

MOV ST\_0,#1

WRBYTE ST\_0,CB\_ST\_STOR

' Wait for the permutation calculator to complete

cb\_wait5 MOV ST\_0,CB\_ST\_STOR

RDBYTE ST\_1,ST\_0

SUB ST\_1,#2 nr,wz

if\_nz JMP #cb\_wait5

' Transfer the cube permutation encoding from global memory to local

CALL #m\_perm

' Cube permutation calculated

' If no changes are required move onto next COSET

' For G0 this means that there are no flipped edges

SUB ST\_1,G0\_SOL1 nr,wz

if\_nz JMP #cnr\_hr5

JMP #st5\_next

' Convert to human readable format as stored in the SD card lookup

' tables

cnr\_hr5 CALL #make\_hr

' Can be used for debugging

'MOV ST\_3,SPI\_GO

'CALL #st\_s

' Do SPI Lookup

' Based on the encoded cube permutation, lookup the move that will

' take the cube from G0 into somewhere in G1

' The SPI routine writes the result into a shared global memory

' location, the physical cube turner will start as soon as data appears

MOV ST\_0,#1 ' COSET files to look for

MOV ST\_1,SPI\_GO

ADD ST\_1,#1

WRBYTE ST\_0,ST\_1

MOV ST\_0,COSET\_G0\_LM ' The total number of lines in lookup

ADD ST\_1,#3 ' table

WRLONG ST\_0,ST\_1

MOV ST\_0,#1 ' Start the SPI search

MOV ST\_1,SPI\_GO

WRBYTE ST\_0,ST\_1

st\_51 RDBYTE ST\_0,ST\_1 wz' Wait for lookup

if\_nz JMP #st\_51

' Handy for debugging

'MOV ST\_3,CB\_CONFIG

'CALL #st\_s1a

MOV ST\_3,SPI\_RSLT

CALL #st\_s

' Start the virtual cube turner, this will allow us to start to look for the

' next set of moves

MOV ST\_0,#1

MOV ST\_1,VIRT\_GO

WRBYTE ST\_0,ST\_1

st\_52 RDBYTE ST\_0,ST\_1 wz' Wait for virtual cube turner to finish

if\_nz JMP #st\_52

st5\_next MOV ST,#6

JMP #state\_0

'' State 6 - Move the cube from G1 to G2

state\_6 SUB ST,#6 nr,wz

if\_nz JMP #state\_7

' Start the state machine that encodes the cube permutations

MOV ST\_0,#3

WRBYTE ST\_0,CB\_ST\_STOR

' Wait for the permutation calculator to complete

cb\_wait6 MOV ST\_0,CB\_ST\_STOR

RDBYTE ST\_1,ST\_0

SUB ST\_1,#4 nr,wz

if\_nz JMP #cb\_wait6

'MOV ST\_3,CB\_ST\_STOR

'CALL #st\_s

'MOV ST\_0,CB\_ST\_STOR

' Transfer the cube permutation encoding from global memory to local

CALL #m\_perm

' Cube permutation calculated

' If no changes are required move onto next COSET

' For G1 this means that all corners are untwisted and all the middle

' slice edges are all in the correct tetrad

SUB ST\_1,G1\_SOL1 nr,wz

if\_nz JMP #cnr\_hr6

SUB ST\_2,G1\_SOL2 nr,wz

if\_nz JMP #cnr\_hr6

JMP #st6\_next

' Convert to human readable format as stored in the SD card lookup

' tables

cnr\_hr6 CALL #make\_hr

' Can be used for debugging

'MOV ST\_3,SPI\_GO

'CALL #st\_s

' Do SPI Lookup

' Based on the encoded cube permutation, lookup the move that will

' take the cube from G1 into somewhere in G2

' The SPI routine writes the result into a shared global memory

' location, the physical cube turner will start as soon as data appears

MOV ST\_1,SPI\_GO

ADD ST\_1,#1

MOV ST\_0,#2 ' Coset file to look up

WRBYTE ST\_0,ST\_1

MOV ST\_0,COSET\_G1\_LM ' The total number of lines in lookup

ADD ST\_1,#3 ' table

WRLONG ST\_0,ST\_1

MOV ST\_0,#1 ' Start the SPI search

MOV ST\_1,SPI\_GO

WRBYTE ST\_0,ST\_1

st\_61 RDBYTE ST\_0,ST\_1 wz' Wait for lookup

if\_nz JMP #st\_61

' Start the virtual cube turner, this will allow us to start to look for the

' next set of moves

MOV ST\_0,#1

MOV ST\_1,VIRT\_GO

WRBYTE ST\_0,ST\_1

st\_62 RDBYTE ST\_0,ST\_1 wz' Wait for lookup

if\_nz JMP #st\_62

' Can be used for debugging

'MOV ST\_3,CB\_CONFIG

'CALL #st\_s1a

MOV ST\_3,SPI\_RSLT

CALL #st\_s

st6\_next MOV ST,#7

JMP #state\_0

'' State 7 - Move the cube from G2 to G3

state\_7 SUB ST,#7 nr,wz

if\_nz JMP #state\_8

' Start the state machine that encodes the cube permutations

MOV ST\_0,#5

WRBYTE ST\_0,CB\_ST\_STOR

' Wait for the permutation calculator to complete

cb\_wait7 MOV ST\_0,CB\_ST\_STOR

RDBYTE ST\_1,ST\_0

SUB ST\_1,#6 nr,wz

if\_nz JMP #cb\_wait7

'MOV ST\_3,CB\_ST\_STOR

'CALL #st\_s

'MOV ST\_0,CB\_ST\_STOR

' Transfer the cube permutation encoding from global memory to local

CALL #m\_perm

' Cube permutation calculated

' If no changes are required move onto next COSET

' For G2 this means that all corners and all the edges

' are all in the correct tetrad

SUB ST\_2,G2\_SOL2 nr,wz

if\_nz JMP #cnr\_hr7

' G2 differs from the others in that it's solved state is expressed

' as one of 96 possible initial states. This comes from the fact that

' while one of the corner tetrads can be in any configuration the other

' is forced into only one of four possible states

' The second, fourth, fifth and seventh must store (any combination) of

' 1, 2 , 3, 4, if not then the cube is not G2 solved

MOV ST\_4,ST\_1

AND ST\_4,G2\_TST\_1

SUB ST\_4,G2\_TST\_2

AND ST\_4,G2\_TST\_3 wz

if\_nz JMP #cnr\_hr7

' Nibbles 1,3,6,8 must be numbers 5, 6, 7 and 8 but only in particular

' combinations. For any given configuration of 1,2,3,4 5,6,7,8 can only

' exist is one of 1 four combinations if the cube is ready to move on to

' the next (G2 to G3).

' Nibbles 6 and 8, 1 and 3, 5 and 7, and 2 and 4 are paired. The pairings

' of the numbers 5,6,7,8 is determined by the positions of the numbers 1,2,3,4. Once

' pairings are determined then the four acceptable combinations are the four

' possible (paired) combinations of 5,6,7,8 in the paired nibbles

' If numbers 1 and 2 are in nibbles 2 and 4 or 5 and 7 then the numbers 5 and 6 are paired and

' 7 and 8 are paired

' If numbers 1 and 3 are in nibbles 2 and 4 or 5 and 7 then the numbers 5 and 7 are paired and

' 6 and 8 are paired

' If numbers 1 and 4 are in nibbles 2 and 4 or 5 and 7 then the numbers 5 and 8 are paired and

' 6 and 7 are paired

MOV ST\_4,ST\_1

SHR ST\_4,#4 ' look at nibble 2

AND ST\_4,#$F

MOV ST\_5,ST\_1

SHR ST\_5,#12 ' look at nibble 4

AND ST\_5,#$F

MOV ST\_6,ST\_1

AND ST\_6,#$F ' look at nibble 1

MOV ST\_7,ST\_1

SHR ST\_7,#8

AND ST\_7,#$F ' look at nibble 3

ADD ST\_6,ST\_7 ' this sum will be used later

MOV ST\_7,ST\_1

SHR ST\_7,#20

AND ST\_7,#$F ' look at nibble 6

MOV ST\_8,ST\_1

SHR ST\_8,#28

AND ST\_8,#$F ' look at nibble 8

ADD ST\_7,ST\_8 ' this sum will be used later

SUB ST\_4,#1 wz ' nibble 2 is 1

if\_z JMP #pr\_1\_X ' determine the other

SUB ST\_4,#1 wz ' nibble 2 is 2

if\_z JMP #pr\_2\_X ' determine the other

SUB ST\_4,#1 wz ' nibble 2 is 3

if\_z JMP #pr\_3\_X ' determine the other

JMP #pr\_4\_X ' niblle 2 is 4, determine the other

' nibble 2 is 1

pr\_1\_X SUB ST\_5,#1

SUB ST\_5,#1 wz ' nibble 4 is 2

if\_z JMP #pr\_1\_2 ' determine what the solved pairings are

SUB ST\_5,#1 wz ' nibble 4 is 3

if\_z JMP #pr\_1\_3 ' determine what the solved pairings are

JMP #pr\_1\_4 ' nibble 4 is 4 determine what the solved pairings are

' nibble 2 is 2

pr\_2\_X SUB ST\_5,#1 wz ' nibble 4 is 1

if\_z JMP #pr\_1\_2 ' determine what the solved pairings are

SUB ST\_5,#1

SUB ST\_5,#1 wz ' nibble 4 is 3 determine what the solved pairings are

if\_z JMP #pr\_2\_3

JMP #pr\_2\_4 ' nibble 4 is 4 determine what the solved pairings are

' nibble 2 is 3

pr\_3\_X SUB ST\_5,#1 wz ' nibble 4 is 1

if\_z JMP #pr\_1\_3 ' determine what the solved pairings are

SUB ST\_5,#1 ' nibble 4 is 2

if\_z JMP #pr\_2\_3 ' determine what the solved pairings are

JMP #pr\_3\_4 ' nibble 4 is 4

' determine what the solved pairings are

' nibble 2 is 4

pr\_4\_X SUB ST\_5,#1 wz' nibble 4 is 1

if\_z JMP #pr\_1\_4 ' determine what the solved pairings are

SUB ST\_5,#1 wz' nibble 4 is 2

if\_z JMP #pr\_2\_4 ' determine what the solved pairings are

JMP #pr\_3\_4 ' nible 4 is 3 determine what the solved pairings are

' Numbers 1 and 2 are paired, numbers 3 and 4 are paired

pr\_1\_2 ' If the numbers 5 and 6 are paired then we're in a solved state

pr\_3\_4 SUB ST\_6,#5 ' Check what was stored in nibbles 1 and 3

SUB ST\_6,#6 wz ' If 6 and 5 are stored here then

if\_z JMP #st7\_next ' the cube is solved

SUB ST\_7,#5 ' If not check what was stored in nibbles 6 and 8

SUB ST\_7,#6 wz ' if 5 and 6 the cube is solved

if\_z JMP #st7\_next

JMP #cnr\_hr7

' Numbers 1 and 3 are paired, numbers 2 and 4 are paired

pr\_1\_3 ' If the numbers 5 and 7 are paired then we're in a solved state

pr\_2\_4 SUB ST\_6,#5 ' Check what was stored in nibbles 1 and 3

SUB ST\_6,#7 wz ' If 7 and 5 are stored here then

if\_z JMP #st7\_next ' the cube is solved

SUB ST\_7,#5 ' If not check what was stored in nibbles 6 and 8

SUB ST\_7,#7 wz ' if 5 and 7 the cube is solved

if\_z JMP #st7\_next

JMP #cnr\_hr7

' Numbers 2 and 3 are paired, numbers 1 and 4 are paired

pr\_2\_3 ' If the numbers 5 and 8 are paired then we're in a solved state

pr\_1\_4 SUB ST\_6,#5 ' Check what was stored in nibbles 1 and 3

SUB ST\_6,#8 wz ' If 8 and 5 are stored here then

if\_z JMP #st7\_next ' the cube is solved

SUB ST\_7,#5 ' If not check what was stored in nibbles 6 and 8

SUB ST\_7,#8 wz ' if 5 and 8 the cube is solved

if\_z JMP #st7\_next

JMP #cnr\_hr7

' Convert to human readable format as stored in the SD card lookup

' tables

cnr\_hr7 CALL #make\_hr

' Do SPI Lookup

' Based on the encoded cube permutation, lookup the move that will

' take the cube from G2 into somewhere in G3

' The SPI routine writes the result into a shared global memory

' location, the physical cube turner will start as soon as data appears

MOV ST\_1,SPI\_GO

ADD ST\_1,#1

MOV ST\_0,#3 ' Coset file to look up

WRBYTE ST\_0,ST\_1

MOV ST\_0,COSET\_G2\_LM ' The total number of lines in lookup

ADD ST\_1,#3 ' table

WRLONG ST\_0,ST\_1

MOV ST\_0,#1 ' Start the SPI search

MOV ST\_1,SPI\_GO

WRBYTE ST\_0,ST\_1

st\_71 RDBYTE ST\_0,ST\_1 wz' Wait for lookup

if\_nz JMP #st\_71

' Start the virtual cube turner, this will allow us to start to look for the

' next set of moves

MOV ST\_0,#1

MOV ST\_1,VIRT\_GO

WRBYTE ST\_0,ST\_1

st\_72 RDBYTE ST\_0,ST\_1 wz' Wait for lookup

if\_nz JMP #st\_72

' Can be used for debugging

'MOV ST\_3,CB\_CONFIG

'CALL #st\_s1a

MOV ST\_3,SPI\_RSLT

CALL #st\_s

st7\_next MOV ST,#8

JMP #state\_0

'' State 8 - Move the cube from G3 to G4

state\_8 SUB ST,#8 nr,wz

if\_nz JMP #state\_9

' Start the state machine that encodes the cube permutations

MOV ST\_0,#7

WRBYTE ST\_0,CB\_ST\_STOR

' Wait for the permutation calculator to complete

cb\_wait8 MOV ST\_0,CB\_ST\_STOR

RDBYTE ST\_1,ST\_0

SUB ST\_1,#8 nr,wz

if\_nz JMP #cb\_wait8

'MOV ST\_3,CB\_ST\_STOR

'CALL #st\_s

'MOV ST\_0,CB\_ST\_STOR

' Transfer the cube permutation encoding from global memory to local

CALL #m\_perm

' Cube permutation calculated

' If no changes are required move onto next COSET

' For G3 this means that the cube is actually solved

SUB ST\_1,G3\_SOL1 nr,wz

if\_nz JMP #cnr\_hr8

SUB ST\_2,G3\_SOL2 nr,wz

if\_nz JMP #cnr\_hr8

SUB ST\_3,G3\_SOL3 nr,wz

if\_nz JMP #cnr\_hr8

JMP #st8\_next

' Convert to human readable format as stored in the SD card lookup

' tables

cnr\_hr8 CALL #make\_hr

' Can be used for debugging

'MOV ST\_3,SPI\_GO

'CALL #st\_s

' Do SPI Lookup

' Based on the encoded cube permutation, lookup the move that will

' take the cube from G3 to the solved state

' The SPI routine writes the result into a shared global memory

' location, the physical cube turner will start as soon as data appears

MOV ST\_1,SPI\_GO

ADD ST\_1,#1

MOV ST\_0,#4 ' Coset file to look up

WRBYTE ST\_0,ST\_1

MOV ST\_0,COSET\_G3\_LM ' The total number of lines in lookup

ADD ST\_1,#3 ' table

WRLONG ST\_0,ST\_1

MOV ST\_0,#1 ' Start the SPI search

MOV ST\_1,SPI\_GO

WRBYTE ST\_0,ST\_1

st\_81 RDBYTE ST\_0,ST\_1 wz' Wait for lookup

if\_nz JMP #st\_81

' Start the virtual cube turner, this isn't really required

' as after this we should already e solved

MOV ST\_0,#1

MOV ST\_1,VIRT\_GO

WRBYTE ST\_0,ST\_1

st\_82 RDBYTE ST\_0,ST\_1 wz' Wait for lookup

if\_nz JMP #st\_82

' Can be used for debugging

'MOV ST\_3,CB\_CONFIG

'CALL #st\_s1a

MOV ST\_3,SPI\_RSLT

CALL #st\_s

st8\_next MOV ST,#9

JMP #state\_0

state\_9 SUB ST,#9 nr,wz

if\_nz JMP #state\_0

' Can be used for debugging

'MOV ST\_3,CB\_CONFIG

'CALL #st\_s1a

st9\_next MOV ST,#9

JMP #state\_0

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' This subroutine writes single commands to the routine that turns

'' the cube and waits until the turning routine is complete

'' Uses \_0 and \_1, no returned values

nxt\_face

MOV ST\_1,SPI\_RSLT ' particular command to achieve

ADD ST\_1,#1

WRBYTE ST\_0,ST\_1

SUB ST\_1,#1

MOV ST\_0,#$1

WRBYTE ST\_0,ST\_1

st1\_wait\_act RDBYTE ST\_0,SPI\_RSLT wz

if\_nz JMP #st1\_wait\_act

nxt\_face\_ret RET

'' Start the routine that will capture the colour info on the face

'' that is oriented towards the camera, then wait for this routine to

'' complete

'' Uses \_0, no returned values

get\_clrs WRBYTE ST\_0,CLR\_CAP\_INFO

st1\_wait\_clr RDBYTE ST\_0,CLR\_CAP\_INFO wz

if\_nz JMP #st1\_wait\_clr

get\_clrs\_ret RET

'' This routine takes the information returned from the cube permutation

'' calculation routine and transfers them to local routines

m\_perm ADD ST\_0,#4

RDLONG ST\_1,ST\_0

ADD ST\_0,#4

RDLONG ST\_2,ST\_0

ADD ST\_0,#4

RDLONG ST\_3,ST\_0

m\_perm\_ret RET

'' Subroutine will convert 4 bits at a time into

'' ASCII format. So 32 bits will be spread over two long

'' words

'' Data is written to globalb memory location

'' The data to be converted to human readable format are the

'' long words that store the cube permutations

'' Passed

'' ST\_1 - word to be stored in W1

'' ST\_2 - word to be stored in W2

'' ST\_3 - word to be stored in W3

'' Uses ST\_4,ST\_5,ST\_6,ST\_7,ST\_8,ST\_9,ST\_10,ST\_11

make\_hr MOV ST\_11,SPI\_GO

ADD ST\_11,#8

MOV ST\_7,#0 ' source are a total of three long words

MOV ST\_8,#ST\_PERM\_W1\_L ' Destination registers

MOV ST\_9,#ST\_1 ' Source Register

MOV ST\_6,#0

hr\_2 MOVS hr\_a,ST\_9 ' Read the first register passed

NOP ' from the cube configuration

hr\_a MOV ST\_4,0 ' analysis section

MOV ST\_10,ST\_4

MOV ST\_5,#0 ' Start from bit 0 of the long word to convert

hr\_1 MOV ST\_4,ST\_10

ROR ST\_4,ST\_5 ' Move to the next nibble

AND ST\_4,#$F

SUB ST\_4,#$A wc,nr

if\_nc ADD ST\_4,#$27 ' Add offset if nibble represents a letter

OR ST\_6,ST\_4

ADD ST\_6,#$30 ' Convert to ASCII

ROR ST\_6,#8 ' Move to next storage register

ADD ST\_5,#4 ' Move to next nibble

SUB ST\_5,#32 wz,nr ' If done reading the word, skip to the next

if\_z JMP #hr\_4

SUB ST\_5,#16 wz,nr ' If done writing the word, skip to the next

if\_nz JMP #hr\_1

MOVD hr\_b,ST\_8 ' Take the human readable value

NOP

hr\_b MOV 0,ST\_6

WRLONG ST\_6,ST\_11 ' and write out to global memory

ADD ST\_11,#4 ' move the write location to the next memory location

ADD ST\_8,#1 ' move to the next local memory location

MOV ST\_6,#0 ' clear the storage register

JMP #hr\_1

hr\_4 MOVD hr\_c,ST\_8 ' Take the human readable value

NOP

hr\_c MOV 0,ST\_6

WRLONG ST\_6,ST\_11 ' and write out to global memory

ADD ST\_11,#4 ' move the write location to the next memory location

ADD ST\_8,#1 ' move to the next local memory location

MOV ST\_6,#0 ' clear the local storage register

ADD ST\_7,#1 ' Check to see if we have read the last source word

SUB ST\_7,#3 wz,nr

if\_z JMP #make\_hr\_ret

ADD ST\_9,#1 ' If not read in the next word

JMP #hr\_2

make\_hr\_ret RET

{

st\_s1 MOV ST\_0,ST\_OUT

ADD ST\_0,#1

MOV ST\_1,#1

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

MOV ST\_1,ST\_6

ADD ST\_6,#$81

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

MOV ST\_1,#36

WRBYTE ST\_1,ST\_0

MOV ST\_4,#9

st\_t\_next RDLONG ST\_1,ST\_3

ADD ST\_0,#1

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_3,#4

SUB ST\_4,#1 wz

if\_nz JMP #st\_t\_next

ADD ST\_0,#1

MOV ST\_1,#2

WRBYTE ST\_1,ST\_0

MOV ST\_0,ST\_OUT

MOV ST\_1,#40

WRBYTE ST\_1,ST\_0

st\_s1\_ret RET

}

{st\_s1a MOV ST\_3,CB\_CONFIG

MOV ST\_4,#3

ne\_lia MOV ST\_0,ST\_OUT

ADD ST\_0,#1

MOV ST\_1,#1

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

MOV ST\_1,#$81

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

MOV ST\_1,#24

WRBYTE ST\_1,ST\_0

MOV ST\_5,#6

st\_1a\_l RDLONG ST\_2,ST\_3

MOV ST\_1,ST\_2

ADD ST\_0,#1

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_3,#4

SUB ST\_5,#1 wz

if\_nz JMP #st\_1a\_l

ADD ST\_0,#1

MOV ST\_1,#2

WRBYTE ST\_1,ST\_0

MOV ST\_0,ST\_OUT

MOV ST\_1,#28

WRBYTE ST\_1,ST\_0

st\_s1a1 RDBYTE ST\_1,ST\_0 wz

if\_nz JMP #st\_s1a1

SUB ST\_4,#1 wz

if\_nz JMP #ne\_lia

st\_s1a\_ret RET

}

'' Writes data from global memory to serial ort buffer

'' then initializes data transfer

'' Send out the data stored in ST\_3

'' Uses ST\_0,ST\_1,ST\_4,ST\_3

st\_s MOV ST\_0,ST\_OUT ' Fill out the buffer before signalling

ADD ST\_0,#1 ' to the writer routine that the packet is complete

MOV ST\_1,#1

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1 ' Message type

MOV ST\_1,#5

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1 ' Number of bytes

MOV ST\_1,#55

WRBYTE ST\_1,ST\_0

MOV ST\_4,#14 ' Number of long words to read from global memory

st\_t\_next RDLONG ST\_1,ST\_3 ' Read the long word and write out a byte

ADD ST\_0,#1 ' at a time

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_0,#1

ROR ST\_1,#8

WRBYTE ST\_1,ST\_0

ADD ST\_3,#4

SUB ST\_4,#1 wz ' Repeat if there are any words left

if\_nz JMP #st\_t\_next

ADD ST\_0,#1 ' Write out the terminating 0x02

MOV ST\_1,#2

WRBYTE ST\_1,ST\_0

MOV ST\_0,ST\_OUT ' Tell the writer how many bytes to send out

MOV ST\_1,#60

WRBYTE ST\_1,ST\_0

st\_s\_ret RET

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Send out the data stored in ST\_3

'st\_ss MOV ST\_0,ST\_OUT

' ADD ST\_0,#1

' MOV ST\_1,#1

' WRBYTE ST\_1,ST\_0

' ADD ST\_0,#1

' MOV ST\_1,#5

' WRBYTE ST\_1,ST\_0

' ADD ST\_0,#1

' MOV ST\_1,#24

' WRBYTE ST\_1,ST\_0

' MOV ST\_4,#6

'st1\_t\_next RDLONG ST\_1,ST\_3

' ADD ST\_0,#1

' WRBYTE ST\_1,ST\_0

' ADD ST\_0,#1

' ROR ST\_1,#8

' WRBYTE ST\_1,ST\_0

' ADD ST\_0,#1

' ROR ST\_1,#8

' WRBYTE ST\_1,ST\_0

' ADD ST\_0,#1

' ROR ST\_1,#8

' WRBYTE ST\_1,ST\_0

' ADD ST\_3,#4

' SUB ST\_4,#1 wz

' if\_nz JMP #st1\_t\_next

' ADD ST\_0,#1

' MOV ST\_1,#2

' WRBYTE ST\_1,ST\_0

' MOV ST\_0,ST\_OUT

' MOV ST\_1,#28

' WRBYTE ST\_1,ST\_0

'st\_ss\_ret RET

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

SPI long @spi\_pro

CAM\_CAP long @cam\_cap\_pro

FC long @fc\_pro

CLR\_CAP long @clr\_cap\_pro

VID\_OP long @vid\_op\_pro

WRITER long @writer\_pro

PRO4\_OP long @pro\_4

'PRO5\_OP long @pro\_5

READER long @reader\_pro

ACTU long @act\_pro

VIRT\_CT long @virt\_pro

ST\_CONT long @state\_cont

STCB long @cb\_st\_cnt

'NDE long @node\_pro

MD\_CTRL long $00004080

CAP\_RT long $4E81

DL long $04000000

TG long $00000000

ST\_OUT long $4040

ST long $00000000

ST\_0 long $00000000

ST\_1 long $00000000

ST\_2 long $00000000

ST\_3 long $00000000

ST\_4 long $00000000

ST\_5 long $00000000

ST\_6 long $00000000

ST\_7 long $00000000

ST\_8 long $00000000

ST\_9 long $00000000

ST\_10 long $00000000

ST\_11 long $00000000

ST\_12 long $00000000

G0\_SOL1 long $00000000

G1\_SOL1 long $000000F0

G1\_SOL2 long $0000AAAA

G2\_SOL1 long $00000000

G2\_SOL2 long $00001111

G2\_TST\_3 long $0C0CC0C0

G2\_TST\_2 long $01011010

G2\_TST\_1 long $0F0FF0F0

G3\_SOL1 long $12345678

G3\_SOL2 long $12345678

G3\_SOL3 long $00001234

FC\_GO long $0000519C

SPI\_GO long $5D60

VIRT\_GO long $5D00

SPI\_RSLT long $5D80

CB\_CONFIG long $5D20

PARAM\_MASK long $0003FFFF

ST\_PERM\_W1\_L long $00000000

ST\_PERM\_W1\_M long $00000000

ST\_PERM\_W2\_L long $00000000

ST\_PERM\_W2\_M long $00000000

ST\_PERM\_W3\_L long $00000000

ST\_PERM\_W3\_M long $00000000

COSET\_G0\_LM long 2047

COSET\_G1\_LM long 1082564

COSET\_G2\_LM long 2822304

COSET\_G3\_LM long 663551

CLR\_CAP\_INFO long $4E80

CB\_ST\_STOR long $00004084

CLR\_CAP\_DA long $4E84

CLR\_CTR\_FC long $4F5C

PNODE\_PRO long $5260

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Cube Solving State Controller.

'' The cube solver is used to calculate the cube permutations in each of the

'' groups G0,G1,G2,G3

'' The State Controller runs from state to state but is gated by the main state

'' controller

'' States are as follows

'' 0) Idle State

'' 1) Calculate G0 permutation code (look for flipped edges)

'' We are only concerned about the edge flippedness. We

'' encode the position of the cube using 12 bits of an integer

'' 2) Idle State

'' 3) Calculate G1 permutation code (look for twisted corners and middle edges)

'' We are concerned about corner twists and all the

'' middle UD slice edges being in their home tetrad.

'' We encode the position of the cube using

'' a) 12 bits of an integer for the edges

'' b) 16 bits of an integer where pairs of bit position encode

'' the corners twistiness.

'' 4) Idle State

'' 5) Calculate G2 permutation code (look for corners and edges out of tetrads)

'' We are concerned about getting all corner and edges into

'' their home tetrad

'' Tetrad 1 - FL FR BL BR - these were positioned correctly in the G0 to G1

'' Tetrad 2 - FU FD BU BD

'' Tetrad 3 - LU LD RU RD

'' Tetrad 4 - LUF RUB LDB RDF

'' Tetrad 5 - RUF LUB RDB LFD

'' 6) Idle State

'' 7) Calculate G3 permutation code (look for facelets on the other opposite colour face)

'' We are concerned about getting all into their home position

'' 8) Idle State

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

cb\_st\_cnt

'' State 0

'' Idle state, wait for start command

cb\_state\_0 RDBYTE CB\_ST,CB\_ST\_LOOKUP

SUB CB\_ST,#0 nr,wz

if\_nz JMP #cb\_state\_1

JMP #cb\_state\_0

'' State 1

'' The main controller will set the state machine

'' to state 1 when it's ready for G0

cb\_state\_1 RDBYTE CB\_ST,CB\_ST\_LOOKUP

SUB CB\_ST,#1 nr,wz

if\_nz JMP #cb\_state\_2

'' Read in total bytes read in, total number of faces is

'' 54

RDBYTE CBST\_0,CONFIG\_REG wz

SUB CBST\_0,#54 nr,wz

if\_nz JMP #cb\_state\_0

'' Once we have faces, encode G0 permutation

'' Load chief/non-chief face mappings into a scratch area

'' From MSB to LSB uf,ul,ub,ur,lf,lb,rb,rf,df,dl,db,dr

'' Occupies the 12 lowest bits, a one is a flipped edge

'' The solved state is $00000000

'' So binary 00000000000000000000000000000000

MOV CBST\_4,#6 ' Six long words in all

MOV CBST\_5,GX\_LOOKUP ' To be stored in scratch area

MOV CBST\_3,#DAT\_G0G1\_1 ' Table stored locally starting here

CALL #g\_map

'' See if edges are flipped

MOV CBST\_9,#1 ' Signals the bit to be changed

MOV CBST\_5,ED\_FLIP ' Will store the encoded edge flippedness

' All ones indicates all faces flipped

MOV CBST\_11,#0

MOV CBST\_12,#0

MOV CBST\_3,GX\_LOOKUP ' Pointer to the list of chief face colours

g1\_nxt CALL #lk\_ed\_clr

'' If chief is red or orange edge is not flipped

SUB CBST\_8,#$35 nr,wz ' red

if\_z JMP #g1\_clr

SUB CBST\_8,#$36 nr,wz ' orange

if\_z JMP #g1\_clr

'' If chief face is green or blue and the other face

'' if white or yellow edge is not flipped

'' if not then must be red or orange therefore flipped

SUB CBST\_8,#$32 nr,wz ' blue

if\_nz JMP #g1\_tst1

SUB CBST\_7,#$33 nr,wz ' white

if\_z JMP #g1\_clr

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z JMP #g1\_clr

JMP #g1\_dn

g1\_tst1 SUB CBST\_8,#$31 nr,wz ' green

if\_nz JMP #g1\_tst2

SUB CBST\_7,#$33 nr,wz ' white

if\_z JMP #g1\_clr

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z JMP #g1\_clr

g1\_tst2 JMP #g1\_dn

g1\_clr ANDN CBST\_5,CBST\_9

g1\_dn SUB CBST\_9,ED\_MASK wz,nr

if\_z JMP #cb\_st1\_next

SHL CBST\_9,#1 wz

JMP #g1\_nxt

cb\_st1\_next MOV CBST\_3,CB\_ST\_LOOKUP ' write the results to global

ADD CBST\_3,#4 ' memory

WRLONG CBST\_5,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_11,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_12,CBST\_3

MOV CB\_ST,#2 ' Go to state 2 - idle

MOV CBST\_3,CB\_ST\_LOOKUP ' Signal completion to main

WRBYTE CB\_ST,CBST\_3 ' state controller

JMP #cb\_state\_0

'' State 2

'' Once the G0 permutation is encoded wait in state 2 until

'' the next command is received from the main controller

cb\_state\_2 RDBYTE CB\_ST,CB\_ST\_LOOKUP

SUB CB\_ST,#2 nr,wz

if\_nz JMP #cb\_state\_3

cb\_st2\_next 'MOV CB\_ST,#3

JMP #cb\_state\_0

'' Encode G1 permutation

'' Load chief/non-chief face mappings into a scratch area

'' From MSB - uf,ul,ub,ur,lf,lb,rb,rf,df,dl,db,dr - LSB

'' a 1 in a bit position indicates the presence of one

'' of the four middle UD slice edges

'' The solved state is $000000F0

'' So binary 00000000000000000000000011110000

'' The solved state is $0000AAAA

'' So binary 00000000000000001010101010101010

'' 10b is an untwisted corncer

'' 11b if a counterclock wise twisted corner and 01b is a clock

'' wise twisted corner MSB - LUF RUF RDF LDF LUB RUB RDB BDL - LSB

cb\_state\_3 RDBYTE CB\_ST,CB\_ST\_LOOKUP

SUB CB\_ST,#3 nr,wz

if\_nz JMP #cb\_state\_4

MOV CBST\_4,#12 ' Six long words in all

MOV CBST\_5,GX\_LOOKUP ' To be stored in scratch area

MOV CBST\_3,#DAT\_G0G1\_1 ' Table stored locally starting here

CALL #g\_map

MOV CBST\_11,#0

MOV CBST\_12,#0

MOV CBST\_5,#0 ' This is where the human readable

' will be stored

MOV CBST\_9,#1 ' Mask of the bit to be changed

MOV CBST\_3,GX\_LOOKUP ' Pointer to the list of chief face colours

g3\_nxt CALL #lk\_ed\_clr

'' If chief face is green or blue and the other face

'' if white or yellow edge then it belongs in the middle slice

SUB CBST\_8,#$32 nr,wz ' blue

if\_nz JMP #g3\_tst1

SUB CBST\_7,#$33 nr,wz ' white

if\_z JMP #g3\_set

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z JMP #g3\_set

JMP #g3\_dn

g3\_tst1 SUB CBST\_8,#$31 nr,wz ' green

if\_nz JMP #g3\_tst2

SUB CBST\_7,#$33 nr,wz ' white

if\_z JMP #g3\_set

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z JMP #g3\_set

g3\_tst2 JMP #g3\_dn

g3\_set OR CBST\_5,CBST\_9

g3\_dn SUB CBST\_9,ED\_MASK wz,nr

if\_z JMP #cb\_st3\_next1

SHL CBST\_9,#1 wz

JMP #g3\_nxt

cb\_st3\_next1 MOV CBST\_9,#0 ' Loop Counter

MOV CBST\_11,CBST\_5

MOV CBST\_5,#0

MOV CBST\_3,GX\_LOOKUP ' Pointer to the list of chief face colours

ADD CBST\_3,#24

g3a\_nxt CALL #lk\_crn\_clr

SUB CBST\_8,#$35 nr,wz ' red

if\_z JMP #g3a\_set\_chf

SUB CBST\_8,#$36 nr,wz ' orange

if\_z JMP #g3a\_set\_chf

SUB CBST\_7,#$35 nr,wz ' red

if\_z JMP #g3a\_set\_ccw

SUB CBST\_7,#$36 nr,wz ' orange

if\_z JMP #g3a\_set\_ccw

SUB CBST\_10,#$35 nr,wz ' red

if\_z JMP #g3a\_set\_cw

SUB CBST\_10,#$36 nr,wz ' orange

if\_z JMP #g3a\_set\_cw

g3a\_set\_cw OR CBST\_5,#3 ' Set as clockwise twist

JMP #g3a\_dn

g3a\_set\_ccw OR CBST\_5,#1 ' Set as counter clockwise twist

JMP #g3a\_dn

g3a\_set\_chf OR CBST\_5,#2 ' Set as untwisted

JMP #g3a\_dn

g3a\_dn SUB CBST\_9,#7 wz,nr ' go through each corner

if\_z JMP #cb\_st3\_next2

ADD CBST\_9,#1

ROR CBST\_5,#2 ' Twists are stored as a two bit

JMP #g3a\_nxt ' code

cb\_st3\_next2 ROL CBST\_5,#14 ' Rotate the word back to bit 0

MOV CB\_ST,#4 ' Go to idle state 4

MOV CBST\_3,CB\_ST\_LOOKUP ' Store permutation in global memory

WRBYTE CB\_ST,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_11,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_5,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_12,CBST\_3

JMP #cb\_state\_0

'' State 4

'' Once the G1 permutation is encoded wait in state 4 until

'' the next command is received from the main controller

cb\_state\_4 RDBYTE CB\_ST,CB\_ST\_LOOKUP

SUB CB\_ST,#4 nr,wz

if\_nz JMP #cb\_state\_5

cb\_st4\_next 'MOV CB\_ST,#5

JMP #cb\_state\_0

'' Encode G2 permutation

'' Load face mappings into a scratch area

'' Edges are initialized with edges in home tetrad positon.

'' Since the middle slice edges are already in their home tetrad

'' we ignore these. We mark the edges as

'' We need to get these edges into their home tetrads

'' M fbbflrrl L

'' S dduudduu S

'' B B

'' The solved state is $00001111

'' So binary 0000000000000000001000100010001

'' with the edges in the ld,rd,ru,lu (tetrad 3) being the ones

'' and Tetrad 2 being the zeros

'' CORNERS

'' M r l l r r l l r L

'' S d d d d u u u u S

'' B f f b b f f b b B

'' where the number in Tetrad 4 are 8-5 and the numbers

'' in Tetrad 5 are 1-4

'' There is no single solution for thse solved permutation but 96 possible

'' initial states

'' The only available moves F2,B2,R2,L2,U2,D2

'' For either corner tetrad if you pick a fixed combination, only four

'' combinations of the the other tetrad are reachable.

'' the reachable combinations are as follows: take a face, with one tetrad

'' fixed, the other tetrads pair will either be on that face or the opposite face,

'' still paired and can be swapped on the same face.

'' The routine below looks for all of these possible combinations

cb\_state\_5 SUB CB\_ST,#5 nr,wz

if\_nz JMP #cb\_state\_6

MOV CBST\_4,#10 ' Six long words in all

MOV CBST\_5,GX\_LOOKUP ' To be stored in scratch area

MOV CBST\_3,#DAT\_G2G3\_1 ' Table stored locally starting here

CALL #g\_map

MOV CBST\_12,#0

MOV CBST\_11,#0

MOV CBST\_5,#0 ' This is where the human readable

' will be stored

MOV CBST\_9,#1 ' Signals the bit to be changed

MOV CBST\_3,GX\_LOOKUP ' Pointer to the list of chief face colours

ADD CBST\_3,#24

g5\_nxt CALL #lk\_ed\_clr

'' If non-chief face is green or blue and the other face

SUB CBST\_7,#$32 nr,wz ' blue

if\_z JMP #g5\_set

SUB CBST\_7,#$31 nr,wz ' green

if\_z JMP #g5\_set

JMP #g5\_dn

g5\_set OR CBST\_5,#1

g5\_dn SUB CBST\_9,#$80 wz,nr

if\_z JMP #st5\_next1

SHL CBST\_9,#1

ROR CBST\_5,#4

JMP #g5\_nxt

st5\_next1 ROR CBST\_5,#4

MOV CBST\_11,CBST\_5

MOV CBST\_9,#0 ' Loop Counter

MOV CBST\_5,#0

MOV CBST\_3,GX\_LOOKUP ' Pointer to the list of chief face colours

g5a\_nxt CALL #lk\_crn\_clr

'' Chief colour will be eitther red or orange

SUB CBST\_8,#$35 nr,wz ' red

if\_nz JMP #g5a\_tst1

SUB CBST\_7,#$31 nr,wz ' green

if\_z OR CBST\_5,#5

SUB CBST\_7,#$32 nr,wz ' blue

if\_z OR CBST\_5,#6

SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#2

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#1

JMP #g5a\_dn

'' Chief colour is orange

g5a\_tst1 SUB CBST\_7,#$31 nr,wz ' green

if\_z OR CBST\_5,#8

SUB CBST\_7,#$32 nr,wz ' blue

if\_z OR CBST\_5,#7

SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#4

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#3

JMP #g5a\_dn

g5a\_dn SUB CBST\_9,#7 wz,nr

if\_z JMP #cb\_st5\_next2

ADD CBST\_9,#1

ROR CBST\_5,#4

JMP #g5a\_nxt

cb\_st5\_next2 ROL CBST\_5,#28

cb\_st5\_next MOV CB\_ST,#6

MOV CBST\_3,CB\_ST\_LOOKUP

WRBYTE CB\_ST,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_5,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_11,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_12,CBST\_3

'MOV CBST\_3,CB\_ST\_LOOKUP

'CALL #st\_send

JMP #cb\_state\_0

'' State 6

'' Once the G2 permutation is encoded wait in state 6 until

'' the next command is received from the main controller

cb\_state\_6 SUB CB\_ST,#6 nr,wz

if\_nz JMP #cb\_state\_7

cb\_st6\_next 'MOV CB\_ST,#7

JMP #cb\_state\_0

'' State 7

'' Encode G3 permutation

'' We mark the edges as

'' To get to the final solution all edges must be returned to their home position

'' they are already in their home tetrads we just care about their absolute position

'' M f b b f l r r l L

'' S d d u u d d u u S

'' B 1 2 3 4 5 6 7 8 B

'' The solved state is $12345678

'' So binary 00010010001101000101011001111000

'' M f f b b L

'' S l r l r S

'' B 1 2 3 4 B

'' The solved state is $00001234

'' So binary 00000000000000000001001000110100

'' To get to the final solution all corners must be returned to their

'' home position

'' M r l l r r l l r L

'' S d d d d u u u u S

'' B f f b b f f b b B

'' 1 2 3 4 5 6 7 8

'' The solved state is $12345678

'' So binary 00010010001101000101011001111000

cb\_state\_7 SUB CB\_ST,#7 nr,wz

if\_nz JMP #cb\_state\_8

MOV CBST\_11,#0

MOV CBST\_12,#0

MOV CBST\_4,#18 '' Six long words in all

MOV CBST\_5,GX\_LOOKUP '' To be stored in scratch area

MOV CBST\_3,#DAT\_G2G3\_1 '' Table stored locally starting here

CALL #g\_map

MOV CBST\_12,#0

MOV CBST\_11,#0

MOV CBST\_5,#0 '' This is where the human readable

'' will be stored

MOV CBST\_9,#0 '' Signals the bit to be changed

MOV CBST\_3,GX\_LOOKUP '' Pointer to the list of chief face colours

ADD CBST\_3,#24

g7a\_nxt CALL #lk\_ed\_clr

'' Chief colour will be eitther red or orange

SUB CBST\_8,#$35 nr,wz ' red

if\_nz JMP #g7a\_tst1

SUB CBST\_7,#$31 nr,wz ' green

if\_z OR CBST\_5,#7

SUB CBST\_7,#$32 nr,wz ' blue

if\_z OR CBST\_5,#8

SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#4

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#3

JMP #g7a\_dn

'' Chief colour is orange

g7a\_tst1 SUB CBST\_7,#$31 nr,wz ' green

if\_z OR CBST\_5,#6

SUB CBST\_7,#$32 nr,wz ' blue

if\_z OR CBST\_5,#5

SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#1

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#2

JMP #g7a\_dn

g7a\_dn SUB CBST\_9,#7 wz,nr

if\_z JMP #cb\_st7\_next1

ADD CBST\_9,#1

ROR CBST\_5,#4

JMP #g7a\_nxt

cb\_st7\_next1 ROL CBST\_5,#28

MOV CBST\_11,CBST\_5

MOV CBST\_5,#0 '' This is where the human readable

'' will be stored

MOV CBST\_9,#0 '' Signals the bit to be changed

MOV CBST\_3,GX\_LOOKUP '' Pointer to the list of chief face colours

ADD CBST\_3,#48

g7b\_nxt CALL #lk\_ed\_clr

'' Chief colour will be eitther blue or green

SUB CBST\_8,#$32 nr,wz ' blue

if\_nz JMP #g7b\_tst1

SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#1

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#3

JMP #g7b\_dn

'' Chief colour is green

g7b\_tst1 SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#2

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#4

JMP #g7b\_dn

g7b\_dn SUB CBST\_9,#3 wz,nr

if\_z JMP #cb\_st7\_next2

ADD CBST\_9,#1

ROR CBST\_5,#4

JMP #g7b\_nxt

cb\_st7\_next2 ROL CBST\_5,#12

MOV CBST\_12,CBST\_5

MOV CBST\_9,#0 '' Loop Counter

MOV CBST\_5,#0

MOV CBST\_3,GX\_LOOKUP '' Pointer to the list of chief face colours

g7c\_nxt CALL #lk\_crn\_clr

'' Chief colour will be eitther red or orange

SUB CBST\_8,#$35 nr,wz ' red

if\_nz JMP #g7c\_tst1

SUB CBST\_7,#$31 nr,wz ' green

if\_z OR CBST\_5,#8

SUB CBST\_7,#$32 nr,wz ' blue

if\_z OR CBST\_5,#6

SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#5

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#7

JMP #g7c\_dn

'' Chief colour is orange

g7c\_tst1 SUB CBST\_7,#$31 nr,wz ' green

if\_z OR CBST\_5,#1

SUB CBST\_7,#$32 nr,wz ' blue

if\_z OR CBST\_5,#3

SUB CBST\_7,#$33 nr,wz ' white

if\_z OR CBST\_5,#2

SUB CBST\_7,#$34 nr,wz ' yellow

if\_z OR CBST\_5,#4

JMP #g7c\_dn

g7c\_dn SUB CBST\_9,#7 wz,nr

if\_z JMP #cb\_st7\_next3

ADD CBST\_9,#1

ROR CBST\_5,#4

JMP #g7c\_nxt

cb\_st7\_next3 ROL CBST\_5,#28

cb\_st7\_next MOV CB\_ST,#8

MOV CBST\_3,CB\_ST\_LOOKUP

WRBYTE CB\_ST,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_5,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_11,CBST\_3

ADD CBST\_3,#4

WRLONG CBST\_12,CBST\_3

JMP #cb\_state\_0

'' State 8

'' Once the G3 permutation is encoded wait in state 8 until

'' the next command is received from the main controller

cb\_state\_8 SUB CB\_ST,#8 nr,wz

if\_nz JMP #cb\_state\_0

cb\_st8\_next 'MOV CB\_ST,#0

JMP #cb\_state\_0

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' I don't know how to initialize a table into global memory

'' This maps data from the local memory of the cog to global memory

'' CBST\_3 Contains the place to read from

'' CBST\_4 contains the number of words to read

'' CBST\_5 is the place to write to

'' CBST\_6 is scratch

g\_map MOVS g\_map1,CBST\_3 ' Move this address into the source

NOP

g\_map1 MOV CBST\_6,0 ' of this move instruction and read the value

WRLONG CBST\_6,CBST\_5 ' Write this long out to global memory

ADD CBST\_3,#1 ' Increment locaL pointer

ADD CBST\_5,#4 ' Increment global pointer

SUB CBST\_4,#1 wz ' Decrement count

if\_nz JMP #g\_map ' Keep going until all are loaded

g\_map\_ret RET

'' Look up edge colours

'' Uses CBST\_4

'' CBST\_8 - has the chief colur

'' CBST\_7 - has the non-chief

'' CBST\_3 - is a generic pointer

lk\_ed\_clr ADD CBST\_3,#12 ' Non-chiefs are first in the list

RDBYTE CBST\_4,CBST\_3 ' Read what the offset into cube config is

MOV CBST\_6,CONFIG\_REG ' Point to the start of the cube configuration

ADD CBST\_6,#1 ' Skip the first byte which is number of faces

ADD CBST\_6,CBST\_4

RDBYTE CBST\_8,CBST\_6 ' Read the chief face colour

SUB CBST\_3,#12 ' Now read chief face colour

RDBYTE CBST\_4,CBST\_3 ' Read what the offset into cube config is

MOV CBST\_6,CONFIG\_REG ' Point to the start of the cube configuration

ADD CBST\_6,#1 ' Skip the first byte which is number of faces

ADD CBST\_6,CBST\_4

RDBYTE CBST\_7,CBST\_6 ' Read the non-chief face colour

ADD CBST\_3,#1

lk\_ed\_clr\_ret RET

'' Look up corner colours

'' CBST\_10 - has the non-chief (cw)

'' CBST\_8 - has the chief colour

'' CBST\_7 - has the non-chief (ccw)

'' CBST\_3 - is a generic pointer

lk\_crn\_clr ADD CBST\_3,#8 ' Chiefs are in second set

RDBYTE CBST\_4,CBST\_3 ' Read what the offset into cube config is

MOV CBST\_6,CONFIG\_REG ' Point to the start of the cube configuration

ADD CBST\_6,#1 ' Skip the first byte which is number of faces

ADD CBST\_6,CBST\_4

RDBYTE CBST\_8,CBST\_6 ' Read the chief face colour

ADD CBST\_3,#8 ' Now read non-chief face colour that is cw

RDBYTE CBST\_4,CBST\_3 ' Read what the offset into cube config is

MOV CBST\_6,CONFIG\_REG ' Point to the start of the cube configuration

ADD CBST\_6,#1 ' Skip the first byte which is number of faces

ADD CBST\_6,CBST\_4

RDBYTE CBST\_10,CBST\_6 ' Read the non-chief face colour

SUB CBST\_3,#16 ' Now read non-chief face colour that is ccw

RDBYTE CBST\_4,CBST\_3 ' Read what the offset into cube config is

MOV CBST\_6,CONFIG\_REG ' Point to the start of the cube configuration

ADD CBST\_6,#1 ' Skip the first byte which is number of faces

ADD CBST\_6,CBST\_4

RDBYTE CBST\_7,CBST\_6 ' Read the non-chief face colour stored ccw

ADD CBST\_3,#1

lk\_crn\_clr\_ret RET

GX\_LOOKUP long $00004090

CB\_ST\_LOOKUP long $00004084

ED\_FLIP long $00000FFF

ED\_MASK long $00000800

CBST\_OUT long $4040

CB\_ST long $00000000

CBST\_0 long $00000000

CBST\_1 long $00000000

CBST\_2 long $00000000

CBST\_3 long $00000000

CBST\_4 long $00000000

CBST\_5 long $00000000

CBST\_6 long $00000000

CBST\_7 long $00000000

CBST\_8 long $00000000

CBST\_9 long $00000000

CBST\_10 long $00000000

CBST\_11 long $00000000

CBST\_12 long $00000000

CONFIG\_REG long $5D20

COG\_PARAM long $5D64

'' These are index entries into the encode cube representation

'' in $5D20. There are 54 entres, bytes wise in ascii

'' where the values 1 through 6 represent the six faces colours

'' MSB - FU LU UB RU LF LB RB RF FD LD DB DR - LSB

'' Non Chiefs first

DAT\_G0G1\_1 long $19100722

DAT\_G0G1\_2 long $15050317

DAT\_G0G1\_3 long $130A011C

DAT\_G0G1\_4 long $322E3034

DAT\_G0G1\_5 long $0E0C201E

DAT\_G0G1\_6 long $292B2725

'' MSB - LUF RUF RDF LDF LUB RUB RDB LDB - LSB

'' Sorted so that the CCW faces from the chiefs

'' are stored in the first two long words

DAT\_G1\_1 long $021D060f

DAT\_G1\_2 long $0B142118

DAT\_G1\_3 long $2A24332D

DAT\_G1\_4 long $2C26352F

DAT\_G1\_5 long $09002308

DAT\_G1\_6 long $121B1A11

'' MSB - RDF LDF LDB RDB RUF LUF LUB RUB - LSB

DAT\_G2G3\_1 long $140B021D

DAT\_G2G3\_2 long $21180f06

DAT\_G2G3\_3 long $262C2A24

DAT\_G2G3\_4 long $352F2D33

DAT\_G2G3\_5 long $1B120900

DAT\_G2G3\_6 long $1A110823

'' MSB - FD DB UB FU LD DR RU LU- LSB

'' Non Chiefs first

DAT\_G2G3\_7 long $10221C0A

DAT\_G2G3\_8 long $19070113

DAT\_G2G3\_spa long $00000000

DAT\_G2G3\_9 long $2E34252B

DAT\_G2G3\_A long $32302729

DAT\_G2G3\_spb long $00000000

'' Non Chiefs first

'' MSB - FL FR BL BR - LSB

DAT\_G3\_1 long $15170503

DAT\_G3\_spa long $00000000

DAT\_G3\_spb long $00000000

DAT\_G3\_2 long $0E1E0C20

DAT\_G3\_spc long $00000000

DAT\_G3\_spd long $00000000

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' The reader monitors the comm port for communications from

'' a host fixed baud rate of 57,600 8n1

'' Normally Input is high

'' Packet always starts with a 0x01 and ends with a 0x2 these are thrown away

'' 0x01 CMD LNGTH DATA DATA .... 0x2 - each is a byte

'' Most of the code in here is legacy, but is left in place to aid with

'' future debugging

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

reader\_pro

reset\_sm MOV RCV\_ST,#0 ' Start in state 0

MOV NEXT\_PTR2,LIST2 ' This is where data is read from

ADD NEXT\_PTR2,#1 '

MOV LNGTH,#0 ' This is where the read length will ne stored

next\_in MOV DATA\_IN,#0 ' This is where the input data is stored

MOV LOOP\_COUNTER1,#9 ' This is the number of bits per 'byte'

wait\_rec WAITPEQ BIT\_IN,BIT\_IN\_MASK nr ' Wait for input bit (start bit)

MOV TARGET1,CNT ' Once we get start bit

ADD TARGET1,DELAY2 ' State half a bit time counter

WAITCNT TARGET1,DELAY2\_1

loop2 WAITCNT TARGET1,DELAY2\_1 ' Delay a full bit

MOV STOR1,INA ' Copy input bit to temp register

RCL STOR1,#1 wc ' move in data register

RCR DATA\_IN,#1 wc ' Move into storage register

SUB LOOP\_COUNTER1,#1 wz ' read in the remaining bits

if\_nz JMP #loop2

RCR DATA\_IN,#23 ' Once we have all bits rotate to least sig

AND DATA\_IN,#$ff ' byte and mask out and other stuff

rcv\_sm

SUB RCV\_ST,#0 wz,nr ' Based on bytes receuved previously the receiver

if\_z JMP #rcv\_st0 ' will be in a certain state. Jump to the

SUB RCV\_ST,#1 wz,nr ' appropriate state

if\_z JMP #rcv\_st1

SUB RCV\_ST,#2 wz,nr

if\_z JMP #rcv\_st2

SUB RCV\_ST,#3 wz,nr

if\_z JMP #rcv\_st3

JMP #reset\_sm ' After all states, reset the machine for next byte

rcv\_st0 SUB DATA\_IN,#1 wz,nr ' Package should always start with a 1

if\_z ADD RCV\_ST,#1 ' If not keep looking for a 1

JMP #next\_in

rcv\_st1 MOV MODE1,DATA\_IN ' Next byte is data containing the mode or

ADD RCV\_ST,#1 ' command

JMP #next\_in

rcv\_st2 MOV LNGTH,DATA\_IN ' Next byte has the length

MOV WR\_LNGTH,DATA\_IN ' Make a copy for a counter

WRBYTE DATA\_IN,NEXT\_PTR2 ' write this byte to gobal memory

ADD NEXT\_PTR2,#1

ADD RCV\_ST,#1

JMP #next\_in

rcv\_st3 WRBYTE DATA\_IN,NEXT\_PTR2 ' Stay in mode three until we've read all

ADD NEXT\_PTR2,#1 ' the data required by the length

SUB WR\_LNGTH,#1 wz

if\_nz JMP #next\_in

WRBYTE LNGTH,LIST2 ' Once donwe, write the length to the

' the beginning of the global memory area

mode\_set SUB MODE1,#$80 wz,nr ' Once data is recevied, jump to the

if\_z CALL #Mode\_0 ' appropriate routine associated with the

' mode commND

SUB MODE1,#$81 wz,nr

if\_z CALL #Mode\_1

SUB MODE1,#$82 wz,nr

if\_z CALL #Mode\_2

SUB MODE1,#3 wz,nr

if\_z CALL #Mode\_3

SUB MODE1,#4 wz,nr

if\_z CALL #Mode\_4

SUB MODE1,#5 wz,nr

if\_z CALL #Mode\_5

rec\_end JMP #reset\_sm

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' These routines are left in for future debugging puroses

'' Mode 0 - Nothing

Mode\_0

Mode\_0\_ret RET

'' Mode 1 - Start Command from remote host

Mode\_1

MOV STOR1,#1

WRBYTE STOR1,MSTR\_MODE1

Mode\_1\_ret RET

'' Mode 2 - Nothing

Mode\_2

Mode\_2\_ret RET

'' Mode 3 - Nothing

Mode\_3

Mode\_3\_ret RET

'' Mode 4 - Nothing

Mode\_4

Mode\_4\_ret RET

'' Mode 5 - Nothing

Mode\_5

Mode\_5\_ret RET

BIT\_IN long $00000000

BIT\_IN\_MASK long $80000000

DATA\_IN long $00000000

TARGET1 long $00000000

DELAY2 long $0000022B

DELAY2\_1 long $00000457

DELAY3 long $08000000

DELAY3\_1 long $08000000

LOOP\_COUNTER1 long $00000009

LIST2 long $00004000

OUTF1 long $4040

NEXT\_PTR2 long $00004000

RCV\_ST long $00

LNGTH long $00

WR\_LNGTH long $00

MODE1 long $00000000

'CUBE\_CMDS long $00005D00

CUBE\_PAT long $00005D20

CUBE\_PTR1 long $00000000

CUBE\_STOR1 long $00000000

CUBE\_PTR2 long $00000000

CUBE\_STOR2 long $00000000

STOR2\_1 long $00000000

MSTR\_MODE1 long $00004080

STOR1 long $00000000

STOR2 long $00000000

STOR3 long $00000000

STOR4 long $00000000

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' This routine takes centre colour information recorded by the camera and turns it into

'' guesses about all the other facelets

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

fc\_pro RDBYTE STOR18\_0,FC\_CMD wz ' Wait for the start command

if\_z JMP #fc\_pro

'' Generate the average value for each centre face

MOV PTR18\_0,CTR\_FAC ' Where centre face data is stored

MOV PTR18\_1,FC\_DATA ' Where average colour values for centre facelets

' will be stored

MOV LOOPER18\_1,#6 ' Loops through each cube face

fc\_pro\_b MOV STOR18\_0,#0

WRLONG STOR18\_0,PTR18\_1 ' Zero the average value for Y,V,U

MOV STOR18\_4,#0 ' Used to store average Y

MOV STOR18\_5,#0 ' Used to store average V

MOV STOR18\_6,#0 ' Used to store average U

MOV LOOPER18\_0,#24 ' Loop through each of the 24 sampled

' values of facelt colour

fc\_pro\_a RDLONG STOR18\_0,PTR18\_0 ' Colour infor Y,V,U is stored in one long

MOV STOR18\_1,STOR18\_0 ' word with a spare byte

AND STOR18\_1,#$FF ' Extract each byte

SHR STOR18\_0,#8

MOV STOR18\_2,STOR18\_0

AND STOR18\_2,#$FF

SHR STOR18\_0,#8

MOV STOR18\_3,STOR18\_0

AND STOR18\_3,#$FF

ADD STOR18\_4,STOR18\_1 ' Sum each Y component

ADD STOR18\_5,STOR18\_2 ' Sum each V component

ADD STOR18\_6,STOR18\_3 ' Sum each U component

ADD PTR18\_0,#4 ' Skip to the next coour sample

SUB LOOPER18\_0,#1 wz

if\_nz JMP #fc\_pro\_a

MOV STOR18\_1,STOR18\_5 ' Divide V sum by 24

CALL #fc\_div24 ' To calcuate average

MOV STOR18\_2,STOR18\_0

MOV STOR18\_1,STOR18\_6 ' Divide U sum by 24

CALL #fc\_div24 ' to calcuate average

MOV STOR18\_3,STOR18\_0

MOV STOR18\_1,STOR18\_4 ' Divide Y sum by 24

CALL #fc\_div24 ' to calcuate average

MOV STOR18\_1,STOR18\_0

MOV STOR18\_0,STOR18\_3 ' Combine all the averages

SHL STOR18\_0,#8 ' into one word a move

OR STOR18\_0,STOR18\_2

SHL STOR18\_0,#8

OR STOR18\_0,STOR18\_1

WRLONG STOR18\_0,PTR18\_1 ' to global memeory

ADD PTR18\_1,#4 ' move pointer to next memory

' location

SUB LOOPER18\_1,#1 wz ' Move to next face

if\_nz JMP #fc\_pro\_b

'' For each facelet determine which colour it most likely is

'' Write in order into cube solver

MOV PTR18\_0,SCAN\_FAC ' Where the initial cube permutation will

' be stored

MOV PTR18\_2,CLRS ' Where each set of avergae facelt values is

' stored

MOV LOOPER18\_0,#54 ' There are 54 facelets in total, we don't

' have to do the centre ones but will

' anyways

fc\_pro\_c RDLONG STOR18\_0,PTR18\_2 ' Extract each component Y,V,U of the

MOV STOR18\_1,STOR18\_0 ' facelet to be analyzed

SHR STOR18\_1,#8

AND STOR18\_1,#$FF

MOV STOR18\_2,STOR18\_0

SHR STOR18\_2,#16

AND STOR18\_2,#$FF

AND STOR18\_0,#$FF

MOV PTR18\_1,FC\_DATA ' Compare each newly calcualted average centre

CALL #fc\_cc\_dis ' facelets colours with the present facelet

MOV GR\_D,STOR18\_4 ' Calcuate the squared distance to average of 'green'

ADD PTR18\_1,#4

CALL #fc\_cc\_dis

MOV BL\_D,STOR18\_4 ' Calcuate the squared distance to average of 'green'

ADD PTR18\_1,#4

CALL #fc\_cc\_dis

MOV WH\_D,STOR18\_4 ' Calcuate the squared distance to average of 'green'

ADD PTR18\_1,#4

CALL #fc\_cc\_dis

MOV YL\_D,STOR18\_4 ' Calcuate the squared distance to average of 'green'

ADD PTR18\_1,#4

CALL #fc\_cc\_dis

MOV RD\_D,STOR18\_4 ' Calcuate the squared distance to average of 'green'

ADD PTR18\_1,#4

CALL #fc\_cc\_dis

MOV OR\_D,STOR18\_4 ' Calcuate the squared distance to average of 'green'

ADD PTR18\_1,#4

'' See which colour is closest to the present colour

MOV STOR18\_0,#1 ' Assume it's closest to green

MOV STOR18\_1,GR\_D

SUB BL\_D,STOR18\_1 nr,wc ' See if blue is closer

if\_nc JMP #fc\_pro\_d ' If not compare to white

MOV STOR18\_0,#2 ' Assume it's closest to blue

MOV STOR18\_1,BL\_D

fc\_pro\_d SUB WH\_D,STOR18\_1 nr,wc ' See if white is closer

if\_nc JMP #fc\_pro\_e ' If not compare to yellow

MOV STOR18\_0,#3 ' Assume it's closest to yellow

MOV STOR18\_1,WH\_D

fc\_pro\_e SUB YL\_D,STOR18\_1 nr,wc ' See if yellow is closer

if\_nc JMP #fc\_pro\_f ' If not compare to red

MOV STOR18\_0,#4 ' Assume it's closest to red

MOV STOR18\_1,YL\_D

fc\_pro\_f SUB RD\_D,STOR18\_1 nr,wc ' See if red is closer

if\_nc JMP #fc\_pro\_g ' If not compare to orange

MOV STOR18\_0,#5 ' Assume it's closest to orange

MOV STOR18\_1,RD\_D

fc\_pro\_g SUB OR\_D,STOR18\_1 nr,wc ' See if orange is closer

if\_nc JMP #fc\_pro\_h ' If not we're done

MOV STOR18\_0,#6 ' Must be orange

MOV STOR18\_1,OR\_D

fc\_pro\_h ADD STOR18\_0,#$30 ' Encode into ASCII

MOV STOR18\_1,#FC\_LK\_DATA ' Maps facelet colour result into the order

' used by other routines that analyze the

' permuations

MOV STOR18\_2,#54 ' These manipluations are used to

SUB STOR18\_2,LOOPER18\_0 ' calcuate the offset, looper is a count down

MOV STOR18\_3,STOR18\_2

SHR STOR18\_2,#2 ' There are four offsets per long word

ADD STOR18\_1,STOR18\_2 ' Do the lookup in local memory

MOVS fc\_lk\_up,STOR18\_1

NOP

fc\_lk\_up MOV STOR18\_1,0

AND STOR18\_3,#3 ' Extract the partiular byte we're going to

SHL STOR18\_3,#3 ' to look at

SHR STOR18\_1,STOR18\_3 ' Shift the appropriate byte

AND STOR18\_1,#$FF ' Make sure it's just the lower byte

ADD STOR18\_1,SCAN\_FAC ' Calculate the offset in global memory for the

ADD STOR18\_1,#1 ' Resulting value, offset by one as the first byte

WRBYTE STOR18\_0,STOR18\_1 ' stores the number of facelets

ADD PTR18\_2,#4 ' Jump to the next facelet

SUB LOOPER18\_0,#1 wz

if\_nz JMP #fc\_pro\_c

MOV STOR18\_1,#54 ' Signal that the facelets are all complete

WRBYTE STOR18\_1,SCAN\_FAC

fc\_pro\_end JMP #fc\_pro\_end

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Calcuates the square of the distance from Y,V,U data stored in

'' \_0,\_1,\_2 respectively and the average colours pointed to

'' by PRT18\_1

'' \_0,\_1,\_2 are un changed, result is in \_4

fc\_cc\_dis RDLONG STOR18\_4,PTR18\_1

MOV STOR18\_5,STOR18\_4

SHR STOR18\_5,#8

AND STOR18\_5,#$FF

MOV STOR18\_6,STOR18\_4

SHR STOR18\_6,#16

AND STOR18\_6,#$FF

AND STOR18\_4,#$FF

SUBS STOR18\_4,STOR18\_0

ABS STOR18\_4,STOR18\_4

MOV MULT18\_1,STOR18\_4

MOV MULT18\_2,STOR18\_4

CALL #fc\_mult

MOV STOR18\_4,MULT18\_3

SUBS STOR18\_5,STOR18\_1

ABS STOR18\_5,STOR18\_5

MOV MULT18\_1,STOR18\_5

MOV MULT18\_2,STOR18\_5

CALL #fc\_mult

ADD STOR18\_4,MULT18\_3

SUBS STOR18\_6,STOR18\_2

ABS STOR18\_6,STOR18\_6

MOV MULT18\_1,STOR18\_6

MOV MULT18\_2,STOR18\_6

CALL #fc\_mult

ADD STOR18\_4,MULT18\_3

fc\_cc\_dis\_ret RET

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Unsigned 8 bit multiplication enter as \_0 and \_1 and result

'' is in \_2

fc\_mult MOV MULT18\_3,#0

AND MULT18\_1,#$FF

AND MULT18\_2,#$FF

fc\_mult\_b AND MULT18\_1,#1 wz,nr

if\_z JMP #fc\_mult\_a

ADD MULT18\_3,MULT18\_2

fc\_mult\_a SHL MULT18\_2,#1

SHR MULT18\_1,#1 wz

if\_nz JMP #fc\_mult\_b

fc\_mult\_ret RET

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Divides the unsigned integer in \_1 by 24 and stores result

'' in \_0

'' uses \_0 and \_1

fc\_div24 MOV STOR18\_0,#0

SHR STOR18\_1,#2

ADD STOR18\_0,STOR18\_1

SHR STOR18\_1,#2

ADD STOR18\_0,STOR18\_1

SHR STOR18\_1,#2

ADD STOR18\_0,STOR18\_1

SHR STOR18\_1,#2

ADD STOR18\_0,STOR18\_1

SHR STOR18\_1,#2

ADD STOR18\_0,STOR18\_1

SHR STOR18\_0,#3

AND STOR18\_0,#$FF

fc\_div24\_ret RET

SCAN\_FAC long $00005D20

OUT18 long $00004040

CLRS long $00004E84

CTR\_FAC long $00004F5C

FC\_CMD long $0000519C

NODE\_CODE long $00005260

FC\_DATA long $000051A0

GR\_D long $00000000

BL\_D long $00000000

WH\_D long $00000000

YL\_D long $00000000

RD\_D long $00000000

OR\_D long $00000000

LOOPER18\_0 long $00000000

LOOPER18\_1 long $00000000

PTR18\_0 long $00000000

PTR18\_1 long $00000000

PTR18\_2 long $00000000

STOR18\_0 long $00000000

STOR18\_1 long $00000000

STOR18\_2 long $00000000

STOR18\_3 long $00000000

STOR18\_4 long $00000000

STOR18\_5 long $00000000

STOR18\_6 long $00000000

MULT18\_1 long $00000000

MULT18\_2 long $00000000

MULT18\_3 long $00000000

MULT18\_4 long $00000000

STOR18\_0a long $00000000

STOR18\_1a long $00000000

STOR18\_2a long $00000000

STOR18\_3a long $00000000

STOR18\_4a long $00000000

STOR18\_5a long $00000000

FC\_LK\_DATA long $1C23201D

FC\_LK\_DAT1 long $1E1B221F

FC\_LK\_DAT2 long $110E0B21

FC\_LK\_DAT3 long $09100D0A

FC\_LK\_DAT4 long $17140F0C

FC\_LK\_DAT5 long $1916131A

FC\_LK\_DAT6 long $02181512

FC\_LK\_DAT7 long $04010805

FC\_LK\_DAT8 long $06030007

FC\_LK\_DAT9 long $252C2926

FC\_LK\_DAT10 long $27242B28

FC\_LK\_DAT11 long $2D30332A

FC\_LK\_DAT12 long $352E3134

FC\_LK\_DAT13 long $00002F32

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Capture one frame of video and store in the camera memory

'' area. Only one sixthteenth of available data is read from the

'' camera data stream. Every fourth horizonal line is read

'' every second vertical page is read

'' One horizontal line is storde locally.

'' During free horizontal period that line is moved to global memory

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

cam\_cap\_pro ANDN DIRA,HS\_BIT ' Set up IO pins for Horizontal Sync

ANDN DIRA,VS\_BIT ' Veritcal Sync

ANDN DIRA,PCL\_BIT1 ' and the pixel clock

next\_screen RDBYTE CLR\_MODE,CAP\_GOING wz ' Wait for next capture run command

if\_z JMP #next\_screen ' refreshing data, also hold the colour

' component to grab

MOV PTR6\_P3,#DATA6 ' DATA6 points to an local area of storage

' in cog memory

MOV PTR6\_P4,PTR6\_P\_START ' Where raw video data will be stored in global RAM

MOV VS\_CNT,#72 ' 72 vertical lines and 88 horizontal

MOV HS\_CNT,#88 ' columns make up one screen

CMP CLR\_MODE,#3 wc ' For U and V grab the upper IO

WAITPEQ VS\_BIT,VS\_BIT ' Wait for VSYNC to set

WAITPNE VS\_BIT,VS\_BIT ' Wait for VSYNC to clear this skips every

' second vertical page

WAITPEQ VS\_BIT,VS\_BIT ' Wait for VSYNC to set

next\_href WAITPEQ HS\_BIT,HS\_BIT ' Wait for HSYNC to set

if\_c WAITPNE PCL\_BIT1,PCL\_BIT2 ' Wait for pixel clock - for U/V need offset of

' one pixel

next\_pix WAITPEQ PCL\_BIT1,PCL\_BIT2 ' Wait for pixel clock

MOV STOR6\_1,INA ' Grab colour component data

MOVD sm,PTR6\_P3 ' Store it locally

ADD PTR6\_P3,#1 ' move to next memory location

sm MOV 0,STOR6\_1 ' repeat for the 88 horizontal

SUB HS\_CNT,#1 wz ' timing works out to reading every

if\_nz JMP #next\_pix ' fourth pixel

SUB VS\_CNT,#1 wz ' this will need to be repeated

if\_nz JMP #wait\_nxt\_line ' for each of the 72 vertical lines

MOV CLR\_MODE,#0 ' Once done flag back to calling routine

WRBYTE CLR\_MODE,CAP\_GOING ' that this screen is done

JMP #next\_screen

wait\_nxt\_line WAITPNE HS\_BIT,HS\_BIT ' wait for this horizontal line sync to be done

WAITPEQ HS\_BIT,HS\_BIT ' this next line we can skip (only

' need 88)use this time to move

MOV PTR6\_P3,#DATA6 ' the locally stored data to

MOV HS\_CNT,#88 ' global memory, 88 bytes to be moved

next\_move MOVS sm2,PTR6\_P3 ' Read them from local memeory

ADD PTR6\_P3,#1

sm2 MOV STOR6\_1,0

CMP CLR\_MODE,#1 wz ' For U and V grab the upper IO

if\_nz ROR STOR6\_1,#8 ' port

WRBYTE STOR6\_1,PTR6\_P4 ' Write the byte in question

ADD PTR6\_P4,#1 ' repeat for all 88 bytes

SUB HS\_CNT,#1 wz

if\_nz JMP #next\_move

MOV HS\_CNT,#88 ' Prpeare for the next horizontal

MOV PTR6\_P3,#DATA6 ' line

WAITPNE HS\_BIT,HS\_BIT ' We've completred the second line

WAITPEQ HS\_BIT,HS\_BIT ' skipped another

WAITPNE HS\_BIT,HS\_BIT ' Skip two more

WAITPEQ HS\_BIT,HS\_BIT '

WAITPNE HS\_BIT,HS\_BIT '

JMP #next\_href ' Repeat the process until we've

' got a total of 72 horizontal

DDR6 long $0000E0FF

TOGGLE long $10000000

VS\_BIT long $00400000

PCL\_BIT2 long $00000100

PCL\_BIT1 long $00000000

STOR6\_1 long $00000000

HS\_CNT long $00000000

VS\_CNT long $00000000

PTR6\_P\_START long $00006000

PTR6\_P3 long $0

PTR6\_P4 long $0

HS\_BIT long $00800000

CAP\_GOING long $4E81

CLR\_MODE long $00000000

DATA6 res 88

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Captures Y,V,U data from global memory. Captures, 24 pixels from the centre

'' facelt and calcuates the average for the the other 8. Controls the capture routine

'' and which colour component is being grabbed

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' CLR\_OUT is the go signal from the main controller

clr\_cap\_pro RDBYTE FACE16,CLR\_OUT wz ' CLR\_OUT used to start the alogirtm

if\_z JMP #clr\_cap\_pro ' that wll scan the colours associated

' with the facelets 0 indicated that

' no colour scanning should occur. 1-6

' indicate that succsive faces are scanned

' Its value 1 to 6 is used to generate

' offsets into global memory

' where data will be stored

MOV CLR16,#1

nxt\_clr\_cap WRBYTE CLR16,CAP\_RUN ' Command capture to capture Y or U or V

wt\_for\_cap RDBYTE STOR16\_2,CAP\_RUN wz ' Wait until the command clears indicating

if\_nz JMP #wt\_for\_cap ' that the capture routine has completed

MOV TARGET16,CNT ' This is only for show, puts a little delay in

ADD TARGET16,DELAY16 ' each sample to show that the Cube Solver is

WAITCNT TARGET16,DELAY16 ' grabbing three distinct images per face

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

''

'' Cube patches are are the following co-ordiantes

''

'' P O

'' I F

'' X F

'' E S

'' L E

'' T

''

'' +16

'' ------ ------ ------ ------ 16

'' |1,1 | |1,2 | |1,3 |

'' | | | | | | +3(4 pixels)

'' ------ ------ ------ ------ 19 +11

'' ------ ------ ------ ------ 30

'' |2,1 | | 24 | |2,3 |

'' | | |Samp| | | +3(4 pixels)

'' ------ ------ ------ ------ 33 +13

'' ------ ------ ------ ------ 46

'' |3,1 | |3,2 | |3,3 |

'' | | | | | | +3(4 pixels)

'' ------ ------ ------ ------ 49

''

''

'' | | | | | |

'' | | | | | |

'' | | | | | |

'' | | | | | |

'' 26 31 40 45 56 61 PIXEL

'' (6 pixels)(6 pixels)(6 pixels)

'' +26 +5 +9 +5 +11 +5 OFFSET

'''''''''''''''''''''''''''''''''''''''''''''''''''

'' Calcuate and store in global memory the average values of

'' Y,U and V and record all 24X3 values of Y,U and V for

'' the center facelet

'' Future - the centre colours might require more processing

'' on novelty cube but and average works just fine for a standard cube

MOV STOR16\_0,PTR16\_AVG\_FLT ' Calcuate the offset into where

MOV STOR16\_1,FACE16 ' average colour data will be stored

SUB STOR16\_1,#1 ' Face value starts from 1

MOV STOR16\_2,STOR16\_1 ' Now multily by 36, the number

ADD STOR16\_2,STOR16\_1 ' of bytes per face required to store

ADD STOR16\_2,STOR16\_1 ' the average Y U V

MOV STOR16\_1,STOR16\_2

ADD STOR16\_2,STOR16\_1

ADD STOR16\_2,STOR16\_1

MOV STOR16\_1,STOR16\_2

ROL STOR16\_1,#2

ADD STOR16\_0,STOR16\_1 ' Store this memory offset in STOR16\_0

MOV STOR16\_1,PTR16\_FC\_11 ' Calculate the average for face 1,1

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_12 ' Calculate the average for face 1,2

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_13 ' Calculate the average for face 1,3

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_21 ' Calculate the average for face 2,2

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_22 ' Calculate the average for face 2,2

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_23 ' Calculate the average for face 2.3

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_31 ' Calculate the average for face 3.1

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_32 ' Calculate the average for face 3.2

CALL #cavg\_calc ' Calculate average value

ADD STOR16\_0,#4

MOV STOR16\_1,PTR16\_FC\_33 ' Calculate the average for face 3,3

CALL #cavg\_calc ' Calculate average value

'' Move the centre face data into storage

MOV STOR16\_0,FACE16 ' Take the face value and multiply by 96

SUB STOR16\_0,#1 ' Face starts at one so subract one

MOV STOR16\_1,STOR16\_0

ADD STOR16\_1,STOR16\_0 ' 96 is the number bytes used to store the

ADD STOR16\_1,STOR16\_0 ' YUV data (one long word per pixel so one

ROL STOR16\_1,#5 ' byte per YUV triple is wasted)

ADD STOR16\_1,CLR16 ' Add in tho offset based on YUV element being

SUB STOR16\_1,#1 ' scanned subtract 1 since CLR starts at 1

MOV STOR16\_0,STOR16\_1 ' \_0 Stores where data will be written

ADD STOR16\_0,PTR16\_CTR\_CLR ' Offset into global memory where data is stored

MOV STOR16\_1,PTR16\_FC\_22 ' Load in the location to read data from

MOV STOR16\_5,#0

MOV STOR16\_4,#0

cctr\_x MOV STOR16\_6,STOR16\_1 ' \_1 Stores the location of the start of

ADD STOR16\_6,STOR16\_4 ' global memory associated with this facelet

RDBYTE STOR16\_3,STOR16\_6 ' transfer data between locations

WRBYTE STOR16\_3,STOR16\_0

ADD STOR16\_0,#4 ' Reapeat this for the six pixels in this row

ADD STOR16\_4,#1

SUB STOR16\_4,#6 nr,wz

if\_nz JMP #cctr\_x

MOV STOR16\_4,#0 ' Jump to the next line

ADD STOR16\_1,#88 ' And repeat the 6 pixels for all

ADD STOR16\_5,#1 ' four lines

SUB STOR16\_5,#4 nr,wz

if\_nz JMP #cctr\_x

MOV TARGET16,CNT ' This is only for show

ADD TARGET16,DELAY16

WAITCNT TARGET16,DELAY16

ADD CLR16,#1 ' Goto the next colour componet

AND CLR16,#$3 wz

if\_nz JMP #nxt\_clr\_cap

MOV STOR16\_0,#0 ' If on this face all three colours have been

WRBYTE STOR16\_0,CLR\_OUT

JMP #clr\_cap\_pro ' cycled through put a hold pn this task

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Calcuate the average of the 24 sampled values

cavg\_calc MOV STOR16\_2,#0

MOV STOR16\_5,#0

MOV STOR16\_4,#0

cavg\_x MOV STOR16\_6,STOR16\_1 ' \_1 Stores the location of the start of

ADD STOR16\_6,STOR16\_4 ' global memory associated with this facelet

RDBYTE STOR16\_3,STOR16\_6

ADD STOR16\_2,STOR16\_3

ADD STOR16\_4,#1

SUB STOR16\_4,#6 nr,wz

if\_nz JMP #cavg\_x

MOV STOR16\_4,#0

ADD STOR16\_1,#88

ADD STOR16\_5,#1

SUB STOR16\_5,#4 nr,wz

if\_nz JMP #cavg\_x

ROR STOR16\_2,#5 ' Divide summed value

MOV STOR16\_4,STOR16\_2 ' by approximately 24 to calculate

ROR STOR16\_2,#2 ' average. So divide by 2x2x2 then

ADD STOR16\_4,STOR16\_2 ' consecutive by 4 and use a successive

ROR STOR16\_2,#2 ' appoximation 1/4 + 1/16 + 1/64 ...

ADD STOR16\_4,STOR16\_2

ROR STOR16\_2,#2

ADD STOR16\_4,STOR16\_2

MOV STOR16\_2,STOR16\_4

MOV STOR16\_3,STOR16\_0 ' \_0 hold the long word pointer to where avg

ADD STOR16\_3,CLR16 ' data is stored, add in the byte offset for

SUB STOR16\_3,#1 ' element (Y,U,V) but sub stract one becasue

WRBYTE STOR16\_2,STOR16\_3 ' CLR16 encoding starts at 1, store in global

cavg\_calc\_ret RET ' memory

STOR16\_0 long $00000000

STOR16\_1 long $00000000

STOR16\_2 long $00000000

STOR16\_3 long $00000000

STOR16\_4 long $00000000

STOR16\_5 long $00000000

STOR16\_6 long $00000000

FACE16 long $00000000

CLR16 long $00000000

PTR16\_P\_START long $00006000

PTR16\_FC\_11 long $0000659A '' + 88x16 + 26

PTR16\_FC\_12 long $000065A8 '' + 88x16 + 40

PTR16\_FC\_13 long $000065B8 '' + 88x16 + 56

PTR16\_FC\_21 long $00006A6A '' + 88x30 + 26

PTR16\_FC\_22 long $00006A78 '' + 88x30 + 40

PTR16\_FC\_23 long $00006A88 '' + 88x30 + 56

PTR16\_FC\_31 long $00006FEA '' + 88x46 + 26

PTR16\_FC\_32 long $00006FF8 '' + 88x46 + 40

PTR16\_FC\_33 long $00007008 '' + 88x46 + 56

PTR16\_AVG\_FLT long $4E84

PTR16\_CTR\_CLR long $4F5C

TARGET16 long $0

DELAY16 long $00800000

CLR\_OUT long $4E80

CAP\_RUN long $4E81

OUT16 long $4040

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Serial data sender

'' Fixed baud rate of 57,600 8n1

'' Normally Output is high

'' Data starts at $4040

'' Data Length is at $4040

'' Clear $4040 last to indicate that data was sent

'' Start bit low

'' 8 data bits LSB first

'' Stop bit high

'' One more bit to provide room between bytes

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

writer\_pro OR DIRA,DDR ' Confiugre DIRA as indicated on notes

OR OUTA,BIT\_OUT ' Set serial output to normal high state

wait\_byte RDBYTE DATA\_COUNT,LIST1

TEST DATA\_COUNT,DATA\_COUNT wz,nr ' Test to see if a byte is available

if\_z JMP #wait\_byte ' If its zero just loop and wait

next\_byte ANDN OUTA,BIT\_OUT ' Drop out line for one bit time

MOV TARGET,CNT ' This is the start bit

ADD TARGET,DELAY1

WAITCNT TARGET,DELAY1

ADD NEXT\_PTR,#1 ' Send the byte (LSB first) plus stop

RDBYTE NEXT\_CHAR,NEXT\_PTR

send\_loop AND NEXT\_CHAR,BIT\_TEST wz,nr ' Clear or set bit

if\_nz OR OUTA,BIT\_OUT

AND NEXT\_CHAR,BIT\_TEST wz,nr

if\_z ANDN OUTA,BIT\_OUT

WAITCNT TARGET,DELAY1 ' Wait one bit time

SHR NEXT\_CHAR,#1

SUB BIT\_COUNT,#1 wz

if\_nz JMP #send\_loop ' Set output high for one bit time this is the stop bit

OR OUTA,BIT\_OUT ' Raise the out line for one bit time

MOV TARGET,CNT ' This is the stop bit

ADD TARGET,DELAY1

WAITCNT TARGET,DELAY1

MOV TARGET,CNT ' This is an extra bit

ADD TARGET,DELAY1

WAITCNT TARGET,DELAY1

MOV BIT\_COUNT,#8

SUB DATA\_COUNT,#1 wz

if\_nz JMP #next\_byte

WRBYTE ZERO,LIST1 ' Packet has been sent. Clear the byte so more

MOV NEXT\_PTR,LIST1 ' data can be sent

JMP #wait\_byte ' Jump to wait for next packet

DDR long $4F000000

BIT\_OUT long $40000000

BIT\_COUNT byte $08

DATA\_COUNT long $00000000

TARGET long $00000000

BIT\_TEST long $00000001

DELAY1 long $00000457

LIST1 long $00004040

ZERO long $00000000

NEXT\_PTR long $00004040

NEXT\_CHAR long $00000000

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' This routine gives inforation to the NTSC driver

'' It uses pixel data read from $4100-$4E7F

''

'' 261 lines generated

'' 18 are v-blank

'' ' 6 Hysncs

'' ' 6 inverted Hysncs

'' ' 6 Hysncs

'' 22 lines of black black

'' 221 lines of image data

'' All frames have following sections

'' All frames are 63.5 uS in length

'' 1.5 us of front porch

'' 4.7 us of sync tip

'' 0.6 us of breezeway

'' 2.5 us of colour burst

'' 1.6 us of back porch

'' The remaining is image, 88 x 72 pixels

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

vid\_op\_pro

clr\_vid\_mem

MOV PTR\_COUNT,#0

MOV PTR\_P,PTR\_P\_START ' Begining of pixel data to send to TV

next\_group SUB PTR\_END,PTR\_COUNT wz,nr ' Check to see if we've sent out all the

' data

clr\_vid\_mem\_ret

if\_z JMP #pro\_2\_go ' If not just keep sending out blank data

WRLONG TV\_BLANK\_PIX,PTR\_P

ADD PTR\_P,#4

ADD PTR\_COUNT,#1

JMP #next\_group

pro\_2\_go OR DIRA,DDR\_2 ' See data section for defintions

reset\_vid MOV VCFG,CON\_VCFG

MOV CTRA,CON\_CTRA

MOV FRQA,CON\_FRQA

MOV VSCL,CON\_VSCL

next\_frame

' Lines 0-5 H sync

MOV PTR\_PIXEL,ST\_PTR\_PIXEL ' Send 6 lines of H Sync

MOV V\_H\_UNIT,#0

CALL #equ\_pl

' Lines 6-11 inv hsync ' Send 6 lines of inverted H Sync

CALL #vidn\_sync

CALL #sync\_ln

CALL #vidn\_sync

CALL #sync\_ln

CALL #vidn\_sync

CALL #sync\_ln

CALL #vidn\_sync

CALL #sync\_ln

CALL #vidn\_sync

CALL #sync\_ln

CALL #vidn\_sync

CALL #sync\_ln

' Lines 12-17 hysnc ' Send 6 lines of H Sync

CALL #equ\_pl

MOV V\_V\_LINE,#17

' Lines 18-39 black ' Send 22 lines of Black Screen

frm\_st\_3 MOV V\_H\_UNIT,#0

ADD V\_V\_LINE,#1

SUB V\_V\_LINE,#40 wz,nr ' Check to see if we're at the end of this

if\_z JMP #frm\_st\_4 ' black line section

CALL #vid\_hsync

blank\_line SUB V\_H\_UNIT,#55 wz,nr ' For each of these lines we need to send

if\_z JMP #frm\_st\_3 ' the h sync but with a black line to follow

ADD V\_H\_UNIT,#1 ' 55 units long

MOV VSCL,CON\_VSCL ' Change the bit timing to strech out the video

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK ' output

JMP #blank\_line

' Lines 40-260 image

frm\_st\_4 MOV V\_H\_UNIT,#0 ' Now to send out the image

ADD V\_V\_LINE,#1

SUB V\_V\_LINE,#261 wz,nr

if\_z JMP #reset\_frame

CALL #vid\_hsync ' Need and H Sync for each line

JMP #trans\_line ' Send out one line from memory

reset\_frame MOV V\_V\_LINE,#0 ' Jump to show the next video frame

JMP #next\_frame

trans\_line MOV VSCL,CON\_VSCL2 ' Change to a bit rate to suit the image

MOV TV\_COLOURS,TV\_BASE\_CLR1 ' Use a fixed colour palette (monochrome)

tr\_l RDLONG PIXELS,PTR\_PIXEL ' Read in pixel data

WAITVID TV\_COLOURS,PIXELS ' Put out the pixels

ADD PTR\_PIXEL,#4 ' Next set of pixel data

ADD V\_H\_UNIT,#4 '

SUB V\_H\_UNIT,#54 wz,nr ' Send out 10 x 8 pixels

if\_nz JMP #tr\_l

ADD PTR\_PIXEL,#8 ' Skip the last 8 of 88

ADD RPT\_LINE,#1 ' Repeat this one line four times

AND RPT\_LINE,#3 wz ' If we're repeating a line just

if\_nz SUB PTR\_PIXEL,#48 ' jump back to beginning of line

MOV VSCL,CON\_VSCL ' Reset the bit rate to finish off

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK ' the display area with black

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

JMP #frm\_st\_4

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Sends out the equalization pulses

equ\_pl CALL #vid\_hsync

CALL #blk\_ln

CALL #vid\_hsync

CALL #blk\_ln

CALL #vid\_hsync

CALL #blk\_ln

CALL #vid\_hsync

CALL #blk\_ln

CALL #vid\_hsync

CALL #blk\_ln

CALL #vid\_hsync

CALL #blk\_ln

equ\_pl\_ret RET

'' Sends the horizontal blanking info, including sync and color burst

'' Right now its all monochromatic

vid\_hsync

MOV VSCL,CON\_VSCL1

WAITVID TV\_BASE\_COLOURS,NTSC\_TRANS1

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_SYNC

WAITVID TV\_BASE\_COLOURS,NTSC\_TRANS2

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

ADD V\_H\_UNIT,#14

MOV VSCL,CON\_VSCL

vid\_hsync\_ret RET

'' Sends the inverted horizontal blanking info

vidn\_sync MOV VSCL,CON\_VSCL1

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_SYNC

WAITVID TV\_BASE\_COLOURS,TV\_TRANS\_1

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

WAITVID TV\_BASE\_COLOURS,TV\_TRANS\_2

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_SYNC

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_SYNC

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_SYNC

WAITVID TV\_BASE\_COLOURS,TV\_ALL\_SYNC

ADD V\_H\_UNIT,#14

MOV VSCL,CON\_VSCL

vidn\_sync\_ret RET

'' Sends out a plain black line following the horizontal sync

blk\_ln WAITVID TV\_BASE\_COLOURS,TV\_ALL\_BLACK

ADD V\_H\_UNIT,#1

SUB V\_H\_UNIT,#55 wz,nr ' Test to see if we are in state 0

if\_nz JMP #blk\_ln

MOV V\_H\_UNIT,#0

blk\_ln\_ret RET

'' Sends out a plain sync line

sync\_ln WAITVID TV\_BASE\_COLOURS,TV\_ALL\_SYNC

ADD V\_H\_UNIT,#1

SUB V\_H\_UNIT,#55 wz,nr ' Test to see if we are in state 0

if\_nz JMP #sync\_ln

MOV V\_H\_UNIT,#0

sync\_ln\_ret RET

PTR\_COUNT long 0

PTR\_P\_START long $4100 ' Beginning of pixel data to be sent to video monitor

PTR\_P long $0 ' Moving point to cycle through pixel data

TV\_BLANK\_PIX long $AAAAAAAA ' For generating an all pixels on of a given colour

'TV\_BASE\_CLR long $06060602

TV\_BASE\_CLR1 long $07050402

PTR\_END long 864 ' 12 x 72 the number of pixel bytes per screen

V\_V\_LINE long $00000000 ' Stores the present vertical line 0-261

V\_H\_UNIT long $00000000 ' Stores the logical horz unit in a given line

PIXELS long $00000000

TV\_IMAGE long $FA50FA50 ' For generating an all pixels on of a given colour

TV\_COLOURS long $00000000

TV\_BASE\_COLOURS long $02030200 ' CLR Burts high byte (8Ah works)

TV\_ALL\_BLACK long $55555555 ' For generating all black pixels

TV\_ALL\_SYNC long $00000000 ' For generating all symc

TV\_ALL\_BURST long $FFFFFFFF ' For generating all blue (colour burst?)

TV\_TRANS\_1 long $55555400 ' Used in the transitions periods in the inverted h sync

TV\_TRANS\_2 long $00015555 ' Used in the transitions periods in the inverted h sync

NTSC\_TRANS1 long $00155555 ' Used in the transitions periods in the h sync

NTSC\_TRANS2 long $55400000 ' Used in the transitions periods in the h sync

'NTSC\_TRANS3 long $FFFFFFF5 ' Used in the transitions periods in the h sync

'NTSC\_TRANS4 long $5555557F ' Used in the transitions periods in the h sync

CON\_VCFG long %0101\_0100\_0000\_0000\_0000\_0110\_0000\_1111

' Set VCFG - Baseband to low, 4 colour, chroma,

' use the lower four bits, group 3

CON\_CTRA long %0000\_0111\_0000\_0000\_0000\_0000\_0000\_0000

' Mode 1 and X 8

CON\_FRQA long 480438397 ' freq set at 4 X 3579545

CON\_VSCL long %0000\_0000\_0000\_0000\_0100\_0000\_0100\_0000

' Set VSCL - 4 clocks per pixel, 64 pix per frame

CON\_VSCL1 long %0000\_0000\_0000\_0000\_1000\_0000\_1000\_0000

' Set VSCL - 8 clocks per pixel, 128 pix per frame

CON\_VSCL2 long %0000\_0000\_0000\_0001\_0000\_0001\_0000\_0000

' Set VSCL - 16 clocks per pixel, 256 pix per frame

DDR\_2 long $0F000000

ST\_PTR\_PIXEL long $4280 ' Offset from $4100 to center cube on screen

PTR\_PIXEL long $4100

RPT\_LINE long $0

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Take data read from the camera and map it into pixel data

'' Video signal consists of a display of 88(H) by 72(V) pixels

'' Data from the camera is in a 384(H) by 288(V) format

'' Every fourth pixel is used and every fourth line is used

'' As a rsult some data is left of each horizontal row

'' PROP video hardware is set for two bits per pixel (4 colours)

'' so 16 pixels per 32 bit word. The PROP blits out pixel LSB first.

'' This routine takes the video data read from stored at $6000

'' and reads it in a byte at a time. Each byte presents the magnitude of

'' luminosity of each pixel. Each byte is read and the top two bits

'' are copied as a representative value of magnitude. when these pixels

'' are recorded they are stored reversed. As each pixel is shifted into a

'' temporary register, two extra bits are inserted between each pixel.

'' When eight bits are read and copy, shifted by two bits, is OR'd back into the

'' temporary word - this doubles the width of each pixel. The video word in

'' then stored astarting at $4100 where it will later be read out

'' and displayed on the video monitor. The routine loops through one line

'' (8x11 = 88 pixels)

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

pro\_4

next\_cam MOV PTR4\_CAM,PTR4\_CAM\_START ' Data is read from global video data stored at $6000

MOV CAM4\_COUNT,CAM4\_LENGTH ' Read a total of col\*row/8 bits per (88\*72/8)

MOV PTR4\_P,PTR4\_P\_START ' Store the resultant pixel data starting at $4100

MOV HS4\_COUNT,#88 ' Data is stored in video memory in 8bit words

MOV STOR4\_3,#8 ' Since pixel widths are doubled and each pixel

MOV STOR4\_2,#0 ' is two bits wide, one video word that will be displayed

cam\_pix RDBYTE STOR4\_1,PTR4\_CAM ' only has 8 bits of information in it

RCL STOR4\_1,#25 wc ' If 8 bit word is greater than 127 mark it as a pixel

RCR STOR4\_2,#1 ' PROP setup for 4 colours, take the top two bits

RCL STOR4\_1,#1 wc ' and move these to a temporary word (two bits are

RCR STOR4\_2,#1 ' reversed)

SHR STOR4\_2,#2 ' Pixel 0 needs to be in the lsb of pixel word so

ADD PTR4\_CAM,#1 ' shift them in backwards. Move to the next video

SUB STOR4\_3,#1 wz ' word. Use counter to do this for 8 video words

if\_nz JMP #cam\_pix ' During one iteration the temporar word will be

MOV STOR4\_3,#8 ' shifted 4 bits. 4X8 a total of 32 bits, 2 data and two

MOV STOR4\_1,STOR4\_2 ' blank. Make a copy of this word, shift it over 2 bits

SHL STOR4\_1,#2 ' and or them together, this doubles the bit width to be

OR STOR4\_2,STOR4\_1 ' that will be displayed

WRLONG STOR4\_2,PTR4\_P ' Write the resulting word to video output memory

ADD PTR4\_P,#4 ' Move to the next long word in video memory

SUB HS4\_COUNT,#8 wz ' Count the horizontal pixels in groups of 8

if\_nz JMP #up\_cam\_count ' If we haven't completed on line we need to continue

MOV HS4\_COUNT,#88 ' If we have fiished the line, we need to move to the

ADD PTR4\_P,#4 ' next video word

up\_cam\_count SUB CAM4\_COUNT,#1 wz ' Video words are counted in units of eight pixels

if\_z JMP #next\_cam ' when we've completed a screen we restart on the next

JMP #cam\_pix ' video image

PTR4\_CAM\_START long $6000 ' Raw video data is read from here

PTR4\_CAM long $0 ' Pointer into raw video data

CAM4\_COUNT long $0 ' Used every loop to count down the # of bytes read

CAM4\_LENGTH long 792 ' Total number of bytes per screen

PTR4\_P\_START long $4100 ' Data to be blitted to monitor is stored here

PTR4\_P long $0 ' Pointer into data to be blitted

HS4\_COUNT long 0 ' Used every loop to count down each horz line

STOR4\_1 long 0 ' Generic storage

STOR4\_2 long 0 ' Generic storage

STOR4\_3 long 0 ' Generic storage

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Control cube solver actuators based on solution string

'' Cube string is read from $5D00 to $5D1F

'' $5D00 is reserved fo the toal number of moves

'' Commands are all in ASCII

'' R - right face turn

'' r - right face inverse turn

'' 1 - right face double or or inverse double turn

'' ! - orient right face up

'' L,l,2," - same for left face

'' U,u,3,# - same for up face

'' D,d,4,$ - same for down face

'' F,f,5,% - same for front face

'' B,b,6,& - same for back face

'' Commands represent the face of the cube we have to turn

'' To turn the right face we look for which slot that face is in

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

act\_pro OR DIRA,ARMON ' Set the direction on the IO pins

MOV MOVE\_STATE,#0

CALL #go\_home ' Send the actuators to the home position

next\_fc\_move RDBYTE STOR8\_1,LIST8 wz ' Look for flag that a single commands is in the buffer

if\_z JMP #look\_for\_sol ' these commands are used to re-orient the cube

MOV NEXT\_PTR8,LIST8 ' during colour analysis

ADD NEXT\_PTR8,#1

RDBYTE STOR8\_1,NEXT\_PTR8 ' Get fece to move to

MOV STOR8\_4,STOR8\_1

JMP #make\_mva

look\_for\_sol RDBYTE STOR8\_1,SPI\_SOL wz ' Look for flag that actuators commands are being put

if\_z JMP #next\_fc\_move ' into the multi move buffer. Otherwise jump back to

MOV NEXT\_PTR8,SPI\_SOL ' look for single move

look\_sola RDBYTE STOR8\_1,NEXT\_PTR8 wz ' Idle here if there are no new actuators commands yet

if\_z JMP #look\_sola

MOV STOR8\_4,STOR8\_1

' Process byte

' Convert to face type (without inverse or double turn info)

make\_mva MIN STOR8\_1,#$27 wc,nr ' For Single moves add offset

if\_nc JMP #pro\_byte ' to line up with other face turning commands

ADD STOR8\_1,#$10 ' If signle move, skip processing associated

JMP #slot\_cho ' with face turning moves

pro\_byte MIN STOR8\_1,#$60 wc,nr ' 'Normalize the face turns and non-inverse face

if\_nc SUB STOR8\_1,#$20 ' turns, as the same face will be in the cup for both

SUB STOR8\_1,#$52 nr,wz

if\_z MOV STOR8\_1,#$31

SUB STOR8\_1,#$4C nr,wz

if\_z MOV STOR8\_1,#$32

SUB STOR8\_1,#$55 nr,wz

if\_z MOV STOR8\_1,#$33

SUB STOR8\_1,#$44 nr,wz

if\_z MOV STOR8\_1,#$34

SUB STOR8\_1,#$46 nr,wz

if\_z MOV STOR8\_1,#$35

SUB STOR8\_1,#$42 nr,wz

if\_z MOV STOR8\_1,#$36

' Value read from turn instruction list is now $31-$36

slot\_cho SUB F\_SLOT,STOR8\_1 nr,wz ' Now that command is normalized, we need to determine

if\_z JMP #turn\_f ' which routine to run based on where that face is

SUB B\_SLOT,STOR8\_1 nr,wz ' located

if\_z JMP #turn\_b

SUB U\_SLOT,STOR8\_1 nr,wz

if\_z JMP #turn\_u

SUB D\_SLOT,STOR8\_1 nr,wz

if\_z JMP #turn\_d

SUB L\_SLOT,STOR8\_1 nr,wz

if\_z JMP #turn\_l

SUB R\_SLOT,STOR8\_1 nr,wz

if\_z JMP #turn\_r

' If the face we want is in the back slot, do nothing

turn\_b SUB STOR8\_4,#$27 nr,wc ' Check if this is a re-orient move only

if\_nc JMP #turn\_b0 ' If its already in the back do nothing

JMP #next\_turn

turn\_b0 SUB STOR8\_4,#$3A nr,wc ' Do a single face turn

if\_c CALL #hold\_n\_twost

SUB STOR8\_4,#$60 nr,wc

if\_nc JMP #turn\_b1

CALL #hold\_n\_twost ' Do a double face turn

JMP #next\_turn

turn\_b1 CALL #hold\_n\_twist ' Do an inverse face turn

JMP #next\_turn

' If the face is in the left slot, flip once

turn\_l SUB STOR8\_4,#$27 nr,wc ' Check if this is a re-orient move only

if\_nc JMP #turn\_l0

CALL #flip ' Face is in the left slot flip once

JMP #next\_turn

turn\_l0 CALL #flip ' Face is in the left slot flip once

SUB STOR8\_4,#$3A nr,wc ' Do a single face turn

if\_c CALL #hold\_n\_twost

SUB STOR8\_4,#$60 nr,wc

if\_nc JMP #turn\_l1

CALL #hold\_n\_twost ' Do a double face turn

JMP #next\_turn

turn\_l1 CALL #hold\_n\_twist ' Do an inverse face turn

JMP #next\_turn

' If the face is in the front slot, flip twice

turn\_f SUB STOR8\_4,#$27 nr,wc ' Check if this is a re-orient move only

if\_nc JMP #turn\_f0

CALL #flip ' Face is in the front slot flip twice

CALL #flip

JMP #next\_turn

turn\_f0 CALL #flip ' Face is in the front slot flip twice

CALL #flip

SUB STOR8\_4,#$3A nr,wc ' Do a single face turn

if\_c CALL #hold\_n\_twost

MIN STOR8\_4,#$60 nr,wc

if\_nc JMP #turn\_f1

CALL #hold\_n\_twost ' Do a double face turn

JMP #next\_turn

turn\_f1 CALL #hold\_n\_twist ' Do an inverse face turn

JMP #next\_turn

' If the face is in the right slot, flip three times

turn\_r SUB STOR8\_4,#$27 nr,wc ' Check if this is a re-orient move only

if\_nc JMP #turn\_r0

CALL #flip ' Face is in the right slot flip trice

CALL #flip

CALL #flip

JMP #next\_turn

turn\_r0 CALL #flip ' Face is in the right slot flip trice

CALL #flip

CALL #flip

SUB STOR8\_4,#$3A nr,wc ' Do a single face turn

if\_c CALL #hold\_n\_twost

MIN STOR8\_4,#$60 nr,wc

if\_nc JMP #turn\_r1

CALL #hold\_n\_twost ' Do a double face turn

JMP #next\_turn

turn\_r1 CALL #hold\_n\_twist ' Do an inverse face turn

JMP #next\_turn

' If the face is in the down slot we spin and flip

turn\_d SUB STOR8\_4,#$27 nr,wc ' Check if this is a re-orient move only

if\_nc JMP #turn\_d0

CALL #spin\_n\_flip ' Face is in the down slot spin and flip

JMP #next\_turn

turn\_d0 CALL #spin\_n\_flip ' Face is in the down slot spin and flip

SUB STOR8\_4,#$3A nr,wc ' Do a single face turn

if\_c CALL #hold\_n\_twost

MIN STOR8\_4,#$60 nr,wc

if\_nc JMP #turn\_d1

CALL #hold\_n\_twost ' Do a double face turn

JMP #next\_turn

turn\_d1 CALL #hold\_n\_twist ' Do an inverse face turn

JMP #next\_turn

' if the face is in the up slot we spin and flip, flip

turn\_u SUB STOR8\_4,#$27 nr,wc ' Check if this is a re-orient move only

if\_nc JMP #turn\_u0

CALL #spin\_n\_flip ' Face is in the up slot spin and flip,flip,flip

CALL #flip

CALL #flip

JMP #next\_turn

turn\_u0 CALL #spin\_n\_flip ' Face is in the up slot spin and flip,flip,flip

CALL #flip

CALL #flip

SUB STOR8\_4,#$3A nr,wc ' Do a single face turn

if\_c CALL #hold\_n\_twost

MIN STOR8\_4,#$60 nr,wc

if\_nc JMP #turn\_u1

CALL #hold\_n\_twost ' Do a double face turn

JMP #next\_turn

turn\_u1 CALL #hold\_n\_twist ' Do an inverse face turn

next\_turn RDBYTE STOR8\_1,LIST8 wz ' If we're dong signle turns jump

if\_nz JMP #clr\_sng

ADD NEXT\_PTR8,#1 ' Move to the next command

JMP #look\_sola

clr\_sng MOV STOR8\_1,#0 ' For single, face orientation turns

WRBYTE STOR8\_1,LIST8 ' we need to show that we are done

JMP #next\_fc\_move ' so we can take a picture

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Move servos to home position

go\_home MOV ACTUATOR,ARM1 ' Move Arm 1 to home

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,SERVO ' Move servo to home

MOV DELAY8\_2, DELAY8\_S11

MOV DELAY8\_3, DELAY8\_S12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 2 to home

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A22

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

go\_home\_ret RET

'' Hold the cube and inverse twist the bottom face

hold\_n\_twist

MOV STOR\_SLOT, U\_SLOT ' Reorient the virtual cube

MOV U\_SLOT,L\_SLOT

MOV L\_SLOT,D\_SLOT

MOV D\_SLOT,R\_SLOT

MOV R\_SLOT,STOR\_SLOT

MOV ACTUATOR,SERVO ' Twist the servo

MOV DELAY8\_2, DELAY8\_S11

MOV DELAY8\_3, DELAY8\_S13

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM1 ' Move Arm 1 to hold

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A13

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 2 to hold

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A23

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,SERVO ' Un twist the servo turning the face

MOV DELAY8\_2, DELAY8\_S11

MOV DELAY8\_3, DELAY8\_S12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM1 ' Move Arm 1 to home

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 2 to home

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A22

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

hold\_n\_twist\_ret RET

'' Hold the cube and inverse twist the bottom face

hold\_n\_twost

MOV STOR\_SLOT,U\_SLOT ' Reorient the virtual cube

MOV U\_SLOT,R\_SLOT

MOV R\_SLOT,D\_SLOT

MOV D\_SLOT,L\_SLOT

MOV L\_SLOT,STOR\_SLOT

MOV ACTUATOR,ARM1 ' Move Arm 1 to hold

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A13

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 2 to hold

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A23

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,SERVO ' Twist

MOV DELAY8\_2, DELAY8\_S11

MOV DELAY8\_3, DELAY8\_S13

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#200

CALL #make\_moves

MOV ACTUATOR,ARM1 ' Move Arm 1 to home

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 1 to home

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A22

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,SERVO ' Un twist the servo

MOV DELAY8\_2, DELAY8\_S11

MOV DELAY8\_3, DELAY8\_S12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

hold\_n\_twost\_ret RET

'' Flip the cube

flip

MOV STOR\_SLOT, F\_SLOT ' Reorient the virtual cube

MOV F\_SLOT,R\_SLOT

MOV R\_SLOT,B\_SLOT

MOV B\_SLOT,L\_SLOT

MOV L\_SLOT,STOR\_SLOT

MOV ACTUATOR,ARM1 ' Move Arm 1 to tilt position

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A14

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 2 to shove position

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A23

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM1 ' Move Arm 2 to home

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 1 to home

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A22

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

flip\_ret RET

'' Spin and flip the cube without a twist

spin\_n\_flip

MOV STOR\_SLOT,F\_SLOT ' Reorient the virtual cube

MOV F\_SLOT,U\_SLOT

MOV U\_SLOT,B\_SLOT

MOV B\_SLOT,D\_SLOT

MOV D\_SLOT,STOR\_SLOT

MOV ACTUATOR,SERVO ' Twist the servo

MOV DELAY8\_2, DELAY8\_S11

MOV DELAY8\_3, DELAY8\_S13

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM1 ' Move Arm 1 to tilt

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A14

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 2 to shove

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A23

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM1 ' Move Arm 2 to home

MOV DELAY8\_2, DELAY8\_A11

MOV DELAY8\_3, DELAY8\_A12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,ARM2 ' Move Arm 1 to home

MOV DELAY8\_2, DELAY8\_A21

MOV DELAY8\_3, DELAY8\_A22

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

MOV ACTUATOR,SERVO ' Un twist the servo

MOV DELAY8\_2, DELAY8\_S11

MOV DELAY8\_3, DELAY8\_S12

ADD MOVE\_STATE,#1

MOV MOVE\_PAUSE,#150

CALL #make\_moves

spin\_n\_flip\_ret RET

'' Executes actuators moves

make\_moves SUB MOVE\_PAUSE,#1 wz ' Position command is sent a fixed number of times

if\_z JMP #make\_moves\_ret '

MOV STOR8\_1,Cnt ' Grab the present clock

MOV STOR8\_2,STOR8\_1

MOV STOR8\_3,STOR8\_1

ADD STOR8\_1,DELAY8\_1 ' Add this offset to each move delay

ADD STOR8\_2,DELAY8\_2

ADD STOR8\_3,DELAY8\_2

ADD STOR8\_3,DELAY8\_3

ANDN OUTA,ACTUATOR ' Execute a suitably shaped pulse

WAITCNT STOR8\_2,#0 ' based on delay values stored in Data area

WAITCNT STOR8\_3,#0 ' to move the actuators as desired.

OR OUTA,ACTUATOR

WAITCNT STOR8\_1,#0

JMP #make\_moves

make\_moves\_ret RET

SNG\_MODE long $00000000

SPI\_SOL long $00005D84

LIST\_PTR long $00000000

LIST8 long $00005D80

NEXT\_PTR8 long $0

MOVE\_CNT byte $0

ACT\_OUT long $4040

ADD\_REF long $00000000

' Slots are locations associated with

' the mechanical turning mechanism

' It is initialized with the the front

' of the cube in the back slot of the

' mechanism

' Again stored in ASCII

F\_SLOT long $35 ' Constants that represent the home slot/face

B\_SLOT long $36 '

U\_SLOT long $33 '

D\_SLOT long $34 '

R\_SLOT long $31 '

L\_SLOT long $32 '

STOR\_SLOT long $00 ' Temp storage

DELAY8\_1 long $001387F8 ' Time Delay for 20 ms

DELAY8\_2 long $00000000 ' Time Delay for zero position - variable

DELAY8\_3 long $00000000 ' Time Delay for actual position - variable

ACTUATOR long $00000000 ' Store the actuator to move

MOVE\_COMBO long $00000001

MOVE\_STATE long $00000001

MOVE\_PAUSE long $00000000

DELAY8\_A21 long $00010000 ' Time delay for 1.25ms

DELAY8\_A22 long $3000 ' Variable delay for pulse width - Fully retracted

DELAY8\_A23 long $7D80 ' Variable delay for pulse width - Push to hold

DELAY8\_A24 long $13000 ' Variable delay for pulse width - Push to tilt

DELAY8\_A11 long $00010000 ' Time delay for 1.25ms

DELAY8\_A12 long $0010 ' Variable delay for pulse width - Fully retracted

DELAY8\_A13 long $4000 ' Variable delay for pulse width - Push to hold

DELAY8\_A14 long $AC00 ' Variable delay for pulse width - Push to tilt

DELAY8\_8 long $0009800 ' Time delay for 1.25ms

DELAY8\_9 long $001A000 ' Variable delay for pulse width

DELAY8\_S11 long $00009800 ' Time delay for 1.25ms

DELAY8\_S13 long $4000 ' Untwist the servo

DELAY8\_S12 long $12E00 ' Twist the servo

STOR8\_0 long $00000000

STOR8\_1 long $00000000

STOR8\_2 long $00000000

STOR8\_3 long $00000000

STOR8\_4 long $00000000

'STOR8\_10 long $00000000

'STOR8\_11 long $00000000

'STOR8\_12 long $00000000

'STOR8\_13 long $00000000

'STOR8\_14 long $00000000

ARM1 long $00100000 ' IO pin for ARM1

ARM2 long $00200000 ' IO pin for ARM2

SERVO long $00080000 ' IO pin for SERVO

ARMON long $00380000 ' For DDR

'LEDON8 long $00C00000

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Virtual cube face turner

''

'' The virtual cube turner

'' Data for the virtual cube solver is stored starting at $5D00.

'' The virtual cog solver won't sttart until nonzero

'' data is present. The colour faces are stored in (ascii) format with

'' 3d ($33) - yellow (back)

'' 1d ($31) - blue (left)

'' 2d ($32) - white (front)

'' 0d ($30) - green (right)

'' 4d ($34) - red (up)

'' 5d ($35) - orange (down)

'' the colours of the faces.

'' The faces are stored in the following order

'' B11 B12 B13 B21 B22 B23 B31 B32 B33:L11-L33:F11-F33:R11-R33:U11-U33:D11-D33

'' The face numbering orientation is somewhat arbitrary but each face is

'' -----------

'' | U11 U12 U13 |

'' | U21 U22 U23 |

'' | U31 U32 U33 |

'' ----------- ----------- ----------- -----------

'' | B11 B12 B13 | L11 L12 L13 | F11 F12 F13 | R11 R12 R13 |

'' MAP 1 | B21 B22 B23 | L21 L22 L23 | F21 F22 F23 | R21 R22 R23 |

'' | B31 B32 B33 | L31 L32 L33 | F31 F32 F33 | R31 R32 R33 |

'' ----------- ----------- ----------- -----------

'' | D11 D12 D13 |

'' | D21 D22 D23 |

'' | D31 D32 D33 |

'' -----------

'' with the cube solved B is the yellow face shared with orange D and blue L.

'' 64 bytes are avialable but the only requires 54.

'' Commands, instructions for how to turn the cube, are stored starting at $5D20

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

virt\_pro MOV MOV\_PTR,LIST10 ' Points to the start of the command list

virt\_nxt RDBYTE STOR10\_1,VIRT\_START wz ' Wait for flag to start

if\_z JMP #virt\_nxt

MOV STOR10\_1,#54 ' 54 faces are stored + inital length byte

MOV STOR10\_2,#B11 ' Points to the start of the local memory

MOV STOR10\_3,CUBE\_CONFIG ' will be moved to and from this location in global

ADD STOR10\_3,#1

load\_cube RDBYTE STOR10\_4,STOR10\_3 ' Load the cube configuration from global to local memory

MOVD lc,STOR10\_2

NOP

lc MOV 0,STOR10\_4

ADD STOR10\_2,#1

ADD STOR10\_3,#1

SUB STOR10\_1,#1 wz

if\_nz JMP #load\_cube

look\_nxt\_mv RDBYTE STOR10\_4,MOV\_PTR wz ' Load the next command, if none move back to

if\_z JMP #write\_cube ' global memory

' Process byte

' Convert to face type

' Need to send inverse of coded answer

SUB STOR10\_4,#$52 nr,wz ' Based on command determine face turn to use

if\_z CALL #v\_turn\_\_r ' on virtual cube

SUB STOR10\_4,#$4C nr,wz

if\_z CALL #v\_turn\_\_l

SUB STOR10\_4,#$55 nr,wz

if\_z CALL #v\_turn\_\_u

SUB STOR10\_4,#$44 nr,wz

if\_z CALL #v\_turn\_\_d

SUB STOR10\_4,#$46 nr,wz

if\_z CALL #v\_turn\_\_f

SUB STOR10\_4,#$42 nr,wz

if\_z CALL #v\_turn\_\_b

SUB STOR10\_4,#$72 nr,wz

if\_z CALL #v\_turn\_R

SUB STOR10\_4,#$6C nr,wz

if\_z CALL #v\_turn\_L

SUB STOR10\_4,#$75 nr,wz

if\_z CALL #v\_turn\_U

SUB STOR10\_4,#$64 nr,wz

if\_z CALL #v\_turn\_D

SUB STOR10\_4,#$66 nr,wz

if\_z CALL #v\_turn\_F

SUB STOR10\_4,#$62 nr,wz

if\_z CALL #v\_turn\_B

SUB STOR10\_4,#$31 nr,wz

if\_z CALL #v\_turn\_1

SUB STOR10\_4,#$32 nr,wz

if\_z CALL #v\_turn\_2

SUB STOR10\_4,#$33 nr,wz

if\_z CALL #v\_turn\_3

SUB STOR10\_4,#$34 nr,wz

if\_z CALL #v\_turn\_4

SUB STOR10\_4,#$35 nr,wz

if\_z CALL #v\_turn\_5

SUB STOR10\_4,#$36 nr,wz

if\_z CALL #v\_turn\_6

next\_10\_mv ADD MOV\_PTR,#1 ' Go back to see if there are more moves

JMP #look\_nxt\_mv ' to be made on the virtual cube

'' Write the cube back to global memory

write\_cube MOV STOR10\_1,#54 ' 54 faces are stored

MOV STOR10\_2,#B11 ' They'll be move to local memory

MOV STOR10\_3,CUBE\_CONFIG ' From this location in global

ADD STOR10\_3,#1

w\_cube MOVS wcu,STOR10\_2 ' Modify next instruction to point to

NOP

wcu MOV STOR10\_4,0 ' local register

WRBYTE STOR10\_4,STOR10\_3 ' Write out cube facelet to global memory

ADD STOR10\_2,#1

ADD STOR10\_3,#1

SUB STOR10\_1,#1 wz

if\_nz JMP #w\_cube

MOV STOR10\_1,#0 ' Signal that the routine is done

WRBYTE STOR10\_1,VIRT\_START

JMP #virt\_nxt

'update\_int MOV STOR10\_3,OUTF

' ADD STOR10\_3,#1

' MOV STOR10\_1,#1

' WRBYTE STOR10\_1,STOR10\_3

' ADD STOR10\_3,#1

' MOV STOR10\_2,#4

' WRBYTE STOR10\_2,STOR10\_3

' ADD STOR10\_3,#1

' MOV STOR10\_1,#54

' WRBYTE STOR10\_1,STOR10\_3

' ADD STOR10\_3,#1

' MOV STOR10\_2,#B11

'send\_cube MOVS sc,STOR10\_2 ' Modify next instruction to point to

' NOP

'sc MOV STOR10\_4,0 ' local register

' WRBYTE STOR10\_4,STOR10\_3

' ADD STOR10\_2,#1

' ADD STOR10\_3,#1

' SUB STOR10\_1,#1 wz

' if\_nz JMP #send\_cube

' MOV STOR10\_1,#2

' WRBYTE STOR10\_1,STOR10\_3

' MOV STOR10\_1,#58

' WRBYTE STOR10\_1,OUTF

'virt\_wait JMP #virt\_wait

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Virtual inverse right turn

v\_turn\_\_r

CALL #v\_turn\_R

CALL #v\_turn\_R

CALL #v\_turn\_R

v\_turn\_\_r\_ret RET

'' Virtual inverse left turn

v\_turn\_\_l

CALL #v\_turn\_L

CALL #v\_turn\_L

CALL #v\_turn\_L

v\_turn\_\_l\_ret RET

'' Virtual inverse up turn

v\_turn\_\_u

CALL #v\_turn\_U

CALL #v\_turn\_U

CALL #v\_turn\_U

v\_turn\_\_u\_ret RET

'' Virtual inverse down turn

v\_turn\_\_d

CALL #v\_turn\_D

CALL #v\_turn\_D

CALL #v\_turn\_D

v\_turn\_\_d\_ret RET

'' Virtual inverse front turn

v\_turn\_\_f

CALL #v\_turn\_F

CALL #v\_turn\_F

CALL #v\_turn\_F

v\_turn\_\_f\_ret RET

'' Virtual inverse back turn

v\_turn\_\_b

CALL #v\_turn\_B

CALL #v\_turn\_B

CALL #v\_turn\_B

v\_turn\_\_b\_ret RET

'' Virtual right turn

v\_turn\_R

' Same face

MOV STOR10\_1,R12

MOV STOR10\_2,R13

MOV R13,R11

MOV R12,R21

MOV R11,R31

MOV R21,R32

MOV R31,R33

MOV R32,R23

MOV R33,STOR10\_2

MOV R23,STOR10\_1

' Perpendicular faces

MOV STOR10\_1,U11

MOV STOR10\_2,U12

MOV STOR10\_3,U13

MOV U11,F13

MOV U12,F23

MOV U13,F33

MOV F13,D33

MOV F23,D32

MOV F33,D31

MOV D31,B11

MOV D32,B21

MOV D33,B31

MOV B11,STOR10\_3

MOV B21,STOR10\_2

MOV B31,STOR10\_1

v\_turn\_R\_ret RET

'' Virtual left turn

v\_turn\_L

' Same face

MOV STOR10\_1,L12

MOV STOR10\_2,L13

MOV L13,L11

MOV L12,L21

MOV L11,L31

MOV L21,L32

MOV L31,L33

MOV L32,L23

MOV L33,STOR10\_2

MOV L23,STOR10\_1

' Perpendicular faces

MOV STOR10\_3,U33

MOV STOR10\_2,U32

MOV STOR10\_1,U31

MOV U31,B33

MOV U32,B23

MOV U33,B13

MOV B13,D11

MOV B23,D12

MOV B33,D13

MOV D11,F31

MOV D12,F21

MOV D13,F11

MOV F11,STOR10\_1

MOV F21,STOR10\_2

MOV F31,STOR10\_3

v\_turn\_L\_ret RET

'' Virtual up turn

v\_turn\_U

' Same face

MOV STOR10\_1,U12

MOV STOR10\_2,U13

MOV U13,U11

MOV U12,U21

MOV U11,U31

MOV U21,U32

MOV U31,U33

MOV U32,U23

MOV U33,STOR10\_2

MOV U23,STOR10\_1

' Perpendicular faces

MOV STOR10\_1,B13

MOV STOR10\_2,B12

MOV STOR10\_3,B11

MOV B13,L13

MOV B12,L12

MOV B11,L11

MOV L13,F13

MOV L12,F12

MOV L11,F11

MOV F13,R13

MOV F12,R12

MOV F11,R11

MOV R13,STOR10\_1

MOV R12,STOR10\_2

MOV R11,STOR10\_3

v\_turn\_U\_ret RET

'' Virtual down turn

v\_turn\_D

' Same face

MOV STOR10\_1,D12

MOV STOR10\_2,D13

MOV D13,D11

MOV D12,D21

MOV D11,D31

MOV D21,D32

MOV D31,D33

MOV D32,D23

MOV D33,STOR10\_2

MOV D23,STOR10\_1

' Perpendicular faces

MOV STOR10\_1,F31

MOV STOR10\_2,F32

MOV STOR10\_3,F33

MOV F31,L31

MOV F32,L32

MOV F33,L33

MOV L31,B31

MOV L32,B32

MOV L33,B33

MOV B31,R31

MOV B32,R32

MOV B33,R33

MOV R31,STOR10\_1

MOV R32,STOR10\_2

MOV R33,STOR10\_3

v\_turn\_D\_ret RET

'' Virtual front turn

v\_turn\_F

' Same face

MOV STOR10\_1,F12

MOV STOR10\_2,F13

MOV F13,F11

MOV F12,F21

MOV F11,F31

MOV F21,F32

MOV F31,F33

MOV F32,F23

MOV F33,STOR10\_2

MOV F23,STOR10\_1

' Perpendicular faces

MOV STOR10\_1,U13

MOV STOR10\_2,U23

MOV STOR10\_3,U33

MOV U33,L33

MOV U23,L23

MOV U13,L13

MOV L13,D13

MOV L23,D23

MOV L33,D33

MOV D13,R31

MOV D23,R21

MOV D33,R11

MOV R11,STOR10\_3

MOV R21,STOR10\_2

MOV R31,STOR10\_1

v\_turn\_F\_ret RET

'' Virtual back turn

v\_turn\_B

' Same face

MOV STOR10\_1,B12

MOV STOR10\_2,B13

MOV B13,B11

MOV B12,B21

MOV B11,B31

MOV B21,B32

MOV B31,B33

MOV B32,B23

MOV B33,STOR10\_2

MOV B23,STOR10\_1

' Perpendicular faces

MOV STOR10\_1,U31

MOV STOR10\_2,U21

MOV STOR10\_3,U11

MOV U11,R33

MOV U21,R23

MOV U31,R13

MOV R33,D11

MOV R23,D21

MOV R13,D31

MOV D11,L11

MOV D21,L21

MOV D31,L31

MOV L11,STOR10\_3

MOV L21,STOR10\_2

MOV L31,STOR10\_1

v\_turn\_B\_ret RET

'' Virtual double right turn

v\_turn\_1

CALL #v\_turn\_R

CALL #v\_turn\_R

v\_turn\_1\_ret RET

''Virtual double left turn

v\_turn\_2

CALL #v\_turn\_L

CALL #v\_turn\_L

v\_turn\_2\_ret RET

'' Virtual double up turn

v\_turn\_3

CALL #v\_turn\_U

CALL #v\_turn\_U

v\_turn\_3\_ret RET

'' Virtual double down turn

v\_turn\_4

CALL #v\_turn\_D

CALL #v\_turn\_D

v\_turn\_4\_ret RET

'' Virtual double front turn

v\_turn\_5

CALL #v\_turn\_F

CALL #v\_turn\_F

v\_turn\_5\_ret RET

'' Virtual double back turn

v\_turn\_6

CALL #v\_turn\_B

CALL #v\_turn\_B

v\_turn\_6\_ret RET

STOR10\_1 long $0

STOR10\_2 long $0

STOR10\_3 long $0

STOR10\_4 long $0

MOVE\_CNT10 long $0

NEXT\_PTR10 long $0

OUTF long $4040

LIST10 long $5D84

VIRT\_START long $5D00

CUBE\_CONFIG long $5D20

MOV\_PTR long $0000

'' Store the virtual cube configuration

B11 long $0

B12 long $0

B13 long $0

B21 long $0

B22 long $0

B23 long $0

B31 long $0

B32 long $0

B33 long $0

L11 long $0

L12 long $0

L13 long $0

L21 long $0

L22 long $0

L23 long $0

L31 long $0

L32 long $0

L33 long $0

F11 long $0

F12 long $0

F13 long $0

F21 long $0

F22 long $0

F23 long $0

F31 long $0

F32 long $0

F33 long $0

R11 long $0

R12 long $0

R13 long $0

R21 long $0

R22 long $0

R23 long $0

R31 long $0

R32 long $0

R33 long $0

U11 long $0

U12 long $0

U13 long $0

U21 long $0

U22 long $0

U23 long $0

U31 long $0

U32 long $0

U33 long $0

D11 long $0

D12 long $0

D13 long $0

D21 long $0

D22 long $0

D23 long $0

D31 long $0

D32 long $0

D33 long $0

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

DAT

ORG 0

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Reads data from SD card

'' Data is stored in FAT16 format

'' It is assumed that each sector is 512 bytes

'' First sector will always be the master boot record

'' Bytes 9 and 10 will contain the location in sectors of the first (and

'' assumed only) partition. These bytes are read and the first sector

'' of the partition is read.

'' In the partition boot record a number of entries that are important

'' 0x0D Sectors per cluster

'' 0x0F Number of reserved sectors

'' 0x10 Number of file allocation tables

'' 0x11 Number of entries in root directory

'' 0x16 Number of sectors per FAT

'' The location of the FAT (in sectors) is = (MBR bytes 10:9)+ (PBR byte 0x0D)

'' The start of the directory (in sectors) = Start of FAT + (2\* PBR byte 0x16)

'' From the directory we can find the files we are looking for, the files have

'' been load sequentially, so file 1 is G0toG1LU.txt, file 2 is G0toG1a.txt,

'' file 3 is G1toG2LU.txt, file 4 is G1toG2a.txt, file 5 is G2toG3LU.txt, file 6

'' is G2toG3a.txt, file 7 is G3toG4LU.txt and file 8 is G3toG4a.txt

'' The LU files are stored in the order that they must be read

'' The first 32 bytes of the directory include the volume label skip these

'' The directory file entries are all short file names and are stored in

'' four consectutive 64 byte chunks.

'' The last six bytes store the location in the FAT table of the start of the file

'' and the length of the file

'' The location in the FAT table is in units of clusters, for our purpose

'' we can assume 16 sectors per cluster, also we'll assume that the directory

'' occupies 32 sectors

'' So we know where our first file starts, while were at the directory level we'll

'' figure out where all the files start and store for later.

'' SPI\_OFFSET is the offset (in bits) into the 512 byte data stream

'' where data of interest is located. Four bytes starting at

'' this location is stored in SPI\_DATA1 to 4.

'' The command format is CMD:ARG:CRC

''

'' Stored as

'' :M(34)\_C1(8)\_C2(8)\_C3(8) CRLF

'' - where (X) indicates the number of ASCII characters length

'' M is the inverse move to return the configuration to the next coset

'' C1,C2,C3 are cinfigurations that will be compaered to the existing

'' cube in looking for a match

'' \_ is the ascii character $20 a space

'' length is always 1+34+1+8+1+8+1+8+ CRLF = 64 bytes

'' Data is always read sequentially, one entry at a time

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

spi\_pro

' CLK-out, DI-out, DO-in, CS-out

OR DIRA,DI\_P

OR DIRA,CLK\_P

ANDN DIRA,DO\_P

' Init to CS high Clock low and data high

OR OUTA,DI\_P

ANDN OUTA,CLK\_P

' Clear storage for results

MOV STOR11\_1,#16

MOV STOR11\_2,SPI11\_RD

spi\_clr MOV STOR11\_3,#0

WRLONG STOR11\_3,STOR11\_2

ADD STOR11\_2,#4

SUB STOR11\_1,#1 wz

if\_nz JMP #spi\_clr

' Wake up SD Controller on Card

' Send 80 ones to wake up SD card

' Only need to do this at initialiaztion

MOV TARGET11,CNT ' Add a short delay to ensure

ADD TARGET11,DELAY11 ' everyhting is powered up.

MOV STOR11\_1,ONES11 'SPI\_ARG ' Send a total of

MOV CNT11,#32 ' +32

CALL #spi\_byte

MOV STOR11\_1,ONES11 'SPI\_ARG

MOV CNT11,#32 ' +32

CALL #spi\_byte

MOV STOR11\_1,ONES11 'SPI\_ARG

MOV CNT11,#16 ' + 16 = 80

CALL #spi\_byte

'' Go to idle state - reset the SD card

MOV SPI\_CMD,#$40

MOV SPI\_ARG,#$00

MOV SPI\_CRC,#$95 ' First commands use CRC

CALL #spi\_pck

'' Wait for response

MOV CNT11,#8

CALL #wait\_r1

'' Wait to see if we are in SPI mode - first time through always no

look\_for\_spi MOV STOR11\_2,STOR11\_3 nr,wz

if\_z JMP #in\_spi\_md

'' Go to SPI mode

MOV SPI\_CMD,#$41 '

MOV SPI\_ARG,#$00 '

MOV SPI\_CRC,#$95

CALL #spi\_pck

'' wait for response

MOV CNT11,#8

CALL #wait\_r1

JMP #look\_for\_spi

' Wait for the command that the SPI lookup is required

in\_spi\_md MOV STOR11\_1,SPI11\_GO

RDBYTE STOR11\_2,STOR11\_1

SUB STOR11\_2,#1 wz

if\_nz JMP #in\_spi\_md

'' Now in SPI mode

'' Upto this point its been all initialization and doesn't need

'' to be repeated

'' Load permutation data from memory into local registers

ADD STOR11\_1,#1

RDBYTE SPI\_COSET,STOR11\_1

ADD STOR11\_1,#3

ADD STOR11\_1,#4

RDLONG PERM\_W1\_L,STOR11\_1

ADD STOR11\_1,#4

RDLONG PERM\_W1\_M,STOR11\_1

ADD STOR11\_1,#4

RDLONG PERM\_W2\_L,STOR11\_1

ADD STOR11\_1,#4

RDLONG PERM\_W2\_M,STOR11\_1

ADD STOR11\_1,#4

RDLONG PERM\_W3\_L,STOR11\_1

ADD STOR11\_1,#4

RDLONG PERM\_W3\_M,STOR11\_1

ADD STOR11\_1,#4

'' Depending on which subgroup of the four that is being searched

'' a particular file will need to be opened. SPI\_COSET

'' being non zero indicates that search is a go.

'' The integer value of the SPI\_COSET determines which file

'' to open (see comments above). SPI\_COSET is set externally

'' by processes overseeing the solving of the cube.

'' Once the search is started in a particualr coset, the first

'' thing to do is read from the directory the location of the

'' file. This offset is specified in clusters

wait\_coset MOV SPI\_COSET,SPI\_COSET wz

if\_z JMP #wait\_coset

MOV SPI\_HALF,#0

SUB SPI\_COSET,#1 wz,nr

if\_nz JMP #coset\_lo

ADD SPI\_COSET,#4

coset\_lo CALL #spi\_c\_arg

'' Now we have to search for a matching configuration

'' to the calcuated one.

'' There are two files associated with each COSET

'' One file is the index (lookup - LU ) and the other is the actual COSET

'' look up file.

'' The index file uses one of the cube configuration words

'' as an index into the larger file.

'' The index file has each line in the same format.

'' Each coset list has the same format. The coset files are stored

'' in ASCII to be human readable and need to be converted to the

'' format used internally by the cube solver.

'' When an entry is read all its pattern information as well as

'' the inverse of the move to achieve the position are read and stored.

'' The pattern is compared with the desired pattern, if these match

'' the command process is informed and a move is started.

spi\_rd\_rcds CALL #spi\_rbyte ' Read next 512 bytes from SD Card

MOV STOR11\_3,#SPI\_DATA ' A subset of these will be stored locally

' First config parameter is 36 bytes

ADD STOR11\_3,#9 ' (9 words) in from the start of the record

' one colon plus 34 possible moves + 1 space

spi\_cnt\_rcds MOVS spi\_lk\_wrd1M,STOR11\_3 ' Grab the most sig part of first permutation word

NOP

spi\_lk\_wrd1M MOV COSET\_W1\_M,0

ADD STOR11\_3,#1

MOVS spi\_lk\_wrd1L,STOR11\_3 ' Grab the least sig part of first permutation word

NOP

spi\_lk\_wrd1L MOV COSET\_W1\_L,0

ADD STOR11\_3,#1

MOVS spi\_lk\_wrd2M,STOR11\_3 ' Grab most of the most sig part of second

NOP ' permutation word

spi\_lk\_wrd2M MOV COSET\_W2\_M,0

ADD STOR11\_3,#1

'' There is a space between the first and second permutation word

'' to make the coset files human readable. As a result the most sig word needs to

'' be shifted left by 8, the least needs to shift into the most by 8

'' and the top 8 bits of the third word are shifted into the least sig word

'' For the look up table the second permutation word is the offset into

'' the actual coset file where the permutation (in the first permutation word)

'' starts

MOVS spi\_lk\_wrd2L,STOR11\_3 ' Grab most of the least sig part of

NOP ' second permutation word

spi\_lk\_wrd2L MOV COSET\_W2\_L,0

ADD STOR11\_3,#1

MOVS spi\_lk\_wrd2T,STOR11\_3

NOP

spi\_lk\_wrd2T MOV STOR11\_4,0 ' Grab the remaing bytes of the second permutation

MOV STOR11\_5,#8 ' word

spi\_lk\_wrd2 RCL STOR11\_4,#1 wc ' Shift from the extra word

RCL COSET\_W2\_L,#1 wc ' into the least

RCL COSET\_W2\_M,#1 wc ' into the most

SUB STOR11\_5,#1 wz ' a total of 8 bits

if\_nz JMP #spi\_lk\_wrd2

'' There is a space between the first and second permutation word

'' and another space between the second and the third

'' to make the coset files human readable. As a result the most sig word needs to

'' be shifted left by 16, the least needs to shift into the most by 16

'' and the top 16 bits of the third word are shifted into the least sig word

MOVS spi\_lk\_wrd3M,STOR11\_3 ' Grab half of the least sig word

NOP

spi\_lk\_wrd3M MOV COSET\_W3\_M,0

ADD STOR11\_3,#1

MOVS spi\_lk\_wrd3L,STOR11\_3 ' Grab part of the least and part of the most

NOP ' sig word

spi\_lk\_wrd3L MOV COSET\_W3\_L,0

ADD STOR11\_3,#1

MOVS spi\_lk\_wrd3T,STOR11\_3

NOP

spi\_lk\_wrd3T MOV STOR11\_4,0 ' Grab the remaining bytes of the third perm

MOV STOR11\_5,#16 ' Shift from the extra word

spi\_lk\_wrd3 RCL STOR11\_4,#1 wc

RCL COSET\_W3\_L,#1 wc ' into the least

RCL COSET\_W3\_M,#1 wc ' into the most

SUB STOR11\_5,#1 wz ' a total of 16 bits

if\_nz JMP #spi\_lk\_wrd3

'' LU Offset stores the index entry into the COSET

'' Tables. It has units of lines of COSET entries

'' The G0toG1 LU has zero size

'' This compares the present cube permutation

'' words to the word loaded from the latest line of the lookup

'' file, or if we are looking through coset files, it

'' jumps to compare all the permutations

MOV LU\_OFFSET,#0

SUB SPI\_COSET,#5 wc,nr ' Looking for Coset Files

if\_nc JMP #spi\_noLU ' Not index files

SUB SPI\_COSET,#2 wz,nr ' COSET G1toG2 LU

if\_nz JMP #spi\_LU3

SUB PERM\_W1\_M,COSET\_W1\_M nr,wz ' Check to see if the this permutation

if\_nz JMP #spi\_nxt\_rcd ' is the one we are looking for

SUB PERM\_W1\_L,COSET\_W1\_L nr,wz ' If not jump to the next record

if\_nz JMP #spi\_nxt\_rcd

JMP #spi\_LU\_calc

spi\_LU3 SUB SPI\_COSET,#3 wz,nr ' COSET G2toG3 LU

if\_nz JMP #spi\_LU4

SUB PERM\_W2\_M,COSET\_W1\_M nr,wz ' Check to see if this permutation

if\_nz JMP #spi\_nxt\_rcd ' if the one we are looking for

SUB PERM\_W2\_L,COSET\_W1\_L nr,wz ' If not jump to the next record

if\_nz JMP #spi\_nxt\_rcd

JMP #spi\_LU\_calc

spi\_LU4 SUB PERM\_W3\_M,COSET\_W1\_M nr,wz ' COSET G3toG4 LU. Check to see if this

if\_nz JMP #spi\_nxt\_rcd ' is the permutation we are looking

SUB PERM\_W3\_L,COSET\_W1\_L nr,wz ' for. If not jump to the next record

if\_nz JMP #spi\_nxt\_rcd

spi\_LU\_calc MOV STOR11\_2,#0 ' Calcuate the offset into the COSET file

MOV STOR11\_1,COSET\_W2\_M ' Convert the human readable second permutation

MOV STOR11\_4,#8 ' into an offset. A total of 8 possible

spi\_LU\_lower MOV STOR11\_3,#4 ' digits, 4 digits per coset word

spi\_LU\_nxt ROL STOR11\_1,#8 ' Take the most sig byte

MOV STOR11\_5,STOR11\_1 ' Make a copy

AND STOR11\_5,#$40 nr,wz ' Deals with 'A' to 'F'

if\_nz SUB STOR11\_5,#7

AND STOR11\_5,#$F

ADD STOR11\_2,STOR11\_5 ' Add to total

SUB STOR11\_4,#1 wz ' Look to see if all 8 digits are done

if\_z JMP #spi\_LU\_done

ROL STOR11\_2,#4 ' Shift the bits for the next digit

SUB STOR11\_3,#1 wz ' Look to see if this long word is done

if\_nz JMP #spi\_LU\_nxt

MOV STOR11\_1,COSET\_W2\_L ' Now break down the least sig word

JMP #spi\_LU\_lower

spi\_LU\_done MOV LU\_OFFSET,STOR11\_2 ' Add in the offset found on the lookup table

ANDN LU\_OFFSET,#$7 ' Clear out the lowest bits

ROL LU\_OFFSET,#6 ' scale to offset of bytes

ADD SPI\_COSET,#4

JMP #wait\_coset ' Wait for the next Coset command

'' Here we are looking to find the permutation in the actual coset

spi\_noLU 'CALL #spi\_send

SUB PERM\_W1\_M,COSET\_W1\_M nr,wz ' Look at the first perm most sig word

if\_nz JMP #spi\_nxt\_rcd

SUB PERM\_W1\_L,COSET\_W1\_L nr,wz ' Look at the first perm least sig word

if\_nz JMP #spi\_nxt\_rcd

SUB PERM\_W2\_M,COSET\_W2\_M nr,wz ' Look at the second perm most sig word

if\_nz JMP #spi\_nxt\_rcd

SUB PERM\_W2\_L,COSET\_W2\_L nr,wz ' Look at the second perm least sig word

if\_nz JMP #spi\_nxt\_rcd

SUB PERM\_W3\_M,COSET\_W3\_M nr,wz ' Look at the third perm most sig word

if\_nz JMP #spi\_nxt\_rcd

SUB PERM\_W3\_L,COSET\_W3\_L nr,wz ' Look at the third perm least sig word

if\_nz JMP #spi\_nxt\_rcd

JMP #spi\_show\_perm

spi\_nxt\_rcd MOV STOR11\_5,#SPI\_DATA ' Haven't found a match so we need to look

ADD STOR11\_5,#63 ' up the next record

SUB STOR11\_3,STOR11\_5 nr,wz ' See if we're at the end of the local

if\_nz JMP #spi\_one\_rcd ' memory storage (long words)

MOV STOR11\_1,SPI\_HALF wz ' Local memory is limited, so only half of

if\_nz JMP #spi\_dn\_sec ' of 512 bytes can be stored. This flag

MOV SPI\_HALF,SPI\_FLAG1 ' SPI\_HALF deals with the second half.

MOV SPI\_OFFSET,SPI\_FLAG1 ' If test is not zero we are done

JMP #spi\_rd\_rcds ' If test is zero, we're onto the second half next

spi\_dn\_sec MOV SPI\_HALF,#0 ' Did the second half, clear for next 512 bytes

MOV SPI\_OFFSET,#0

ADD SPI\_ARG,#$1ff ' Jump in units of 512 bytes

ADD SPI\_ARG,#$1 ' , stupid 9 bit limit

JMP #spi\_rd\_rcds

spi\_one\_rcd ADD STOR11\_3,#10 ' Move byte point to start of new record

JMP #spi\_cnt\_rcds

spi\_show\_perm MOV STOR11\_2,SPI11\_RD ' Find where present solution ends

spi\_find\_str RDBYTE STOR11\_1,STOR11\_2 wz ' from global memory

if\_z JMP #spi\_wrt\_out

ADD STOR11\_2,#1 ' haven't found the end yet

JMP #spi\_find\_str ' read next byte

spi\_wrt\_out SUB STOR11\_3,#7 ' Solution for this coset was found

MOV STOR11\_5,#4 ' now write out to global memory, trick

spi\_tst\_nxt MOVS spi\_tst\_wrd,STOR11\_3 ' is we need to write it out backwards

NOP ' as we need to the inverse of this

spi\_tst\_wrd MOV STOR11\_1,0 ' solution

MOV STOR11\_4,#4 ' Go through each byte in the long word

spi\_tst\_byt MOV STOR11\_6,STOR11\_1 ' make a copy

AND STOR11\_6,#$FF ' Just look at the lowest byte

SUB STOR11\_6,#$20 wz ' See if the byte is an ascii space

if\_z JMP #spi\_no\_wr ' if so don't write it out

WRBYTE STOR11\_1,STOR11\_2 ' otherwise write it out

ADD STOR11\_2,#1 ' increment the pointer to global memory

spi\_no\_wr ROR STOR11\_1,#8 ' Move to the next byte

SUB STOR11\_4,#1 wz ' decrment the byte counter (in long word)

if\_nz JMP #spi\_tst\_byt ' Go to the next byte

SUB STOR11\_3,#1 ' Go to next (previous) byte

SUB STOR11\_5,#1 wz

if\_nz JMP #spi\_tst\_nxt

spi\_show\_dn MOV STOR11\_1,#0 ' Found the permutation, so clear

WRBYTE STOR11\_1,SPI11\_GO ' command flag and wait for the next

JMP #in\_spi\_md ' SPI command

spi\_show\_noperm MOV STOR11\_1,#2 ' Did not find the permutation

WRBYTE STOR11\_1,SPI11\_GO ' Should do some more

JMP #in\_spi\_md ' but for now just clear flag and wait

' for next command

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

'' Subroutines

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

spi\_send MOV STOR11\_9,#SPI\_DATA

'ADD STOR11\_9,#16

'MOV SPI\_DATA,SPI\_COSET

MOV STOR11\_2,OUT11\_F

ADD STOR11\_2,#1

MOV STOR11\_1,#1

WRBYTE STOR11\_1,STOR11\_2

ADD STOR11\_2,#1

MOV STOR11\_1,#5

WRBYTE STOR11\_1,STOR11\_2

ADD STOR11\_2,#1

MOV STOR11\_1,#55

WRBYTE STOR11\_1,STOR11\_2

MOV STOR11\_4,#14

spi\_test\_next MOVS spi\_\_d,STOR11\_9

NOP

spi\_\_d MOV STOR11\_1,0

'mov STOR11\_1,SPI\_ARG

ADD STOR11\_2,#1

ROL STOR11\_1,#8

WRBYTE STOR11\_1,STOR11\_2

ADD STOR11\_2,#1

ROL STOR11\_1,#8

WRBYTE STOR11\_1,STOR11\_2

ADD STOR11\_2,#1

ROL STOR11\_1,#8

WRBYTE STOR11\_1,STOR11\_2

ADD STOR11\_2,#1

ROL STOR11\_1,#8

WRBYTE STOR11\_1,STOR11\_2

ADD STOR11\_9,#1

SUB STOR11\_4,#1 wz

if\_nz JMP #spi\_test\_next

ADD STOR11\_2,#1

MOV STOR11\_1,#2

WRBYTE STOR11\_1,STOR11\_2

MOV STOR11\_2,OUT11\_F

MOV STOR11\_1,#60

WRBYTE STOR11\_1,STOR11\_2

spi\_send\_ret RET

'' Calcuates the arguemnt variable used in commands to read data from lookup

'' and coset files, takes the file coset number, the offset related to the LU files

''

''

spi\_c\_arg MOV SPI\_OFFSET,#32 ' skip over the volume info

MOV STOR11\_1,SPI\_COSET

add\_offset SUB STOR11\_1,#1 wz ' skip through each file

if\_nz ADD SPI\_OFFSET,#64 ' entry (64 bytes) until

MOV STOR11\_1,STOR11\_1 wz ' we've got the correct list

if\_nz JMP #add\_offset ' of coset generators

ADD SPI\_OFFSET,#58 ' the 58th and 59th bytes are

MOV SPI\_ARG,SEC\_DIR ' for start of the file

ANDN SPI\_OFFSET,#$1FF wz,nr ' Make sure we're aligned with

if\_nz ADD SPI\_ARG,SPI\_SEC ' 512 byte boundaries

AND SPI\_OFFSET,#$1FF

ROL SPI\_OFFSET,#3 ' the offset in bits

CALL #spi\_rbyte ' Read in the file offset

MOV STOR11\_1,SPI\_DATA ' Reverse bytes for cluster

ROR STOR11\_1,#24

ROR SPI\_DATA,#16 ' word is stored bytewise

AND SPI\_DATA,#$ff ' LSBy,MSBy,X,X

AND STOR11\_1,#$ff ' Make into one word

ROL SPI\_DATA,#8 ' representing the offset in

' in clusters from the

XOR STOR11\_1,SPI\_DATA ' start of the directory

MOV SPI\_DATA,STOR11\_1

ROL STOR11\_1,#4 ' 16 sectors per cluster

ROL STOR11\_1,#9 ' 512 bytes per secotr

MOV SPI\_ARG,SEC\_DIR ' Reset to begining of directory

ADD SPI\_ARG,STOR11\_1 ' this should place us

MOV SPI\_OFFSET,#0 ' at the start of the coset file

SUB SPI\_COSET,#5 nr,wc ' Look to see if we have intervening

if\_c JMP #spi\_c\_arg\_ret ' LU files, if so then use offset

ADD SPI\_ARG,LU\_OFFSET ' as well

spi\_c\_arg\_ret RET

'' Send command to read 512 bytes

'' First the command and respone must occur

'' Unlike other commands the clock with continue to run until 512

'' bytes are returned

spi\_rbyte MOV SPI\_CMD,#$51 ' Send read command

MOV SPI\_CRC,#$00

CALL #spi\_pck

'' wait for response - only 8 bits

MOV CNT11,#8 ' Get response

CALL #wait\_r1

'' wait for data will be 512 btyes

MOV CNT11,#$1FF ' Get the data

ADD CNT11,#1

ROL CNT11,#3

CALL #wait\_d1

spi\_rbyte\_ret RET

'' Command to read 512 bytes has already been sent

'' Need to keep clock running

'' When we see data line drop to zero next bit is the MSB of the first

'' data byte

'' SPI\_OFFSET contains the numbers of bits into the data packet

'' where the desired data sits

'' This routine grabs 4 long word (16 bytes) and stored in SPI\_DATA1 to

'' SPI\_DATA

'' Only 64 long words are actually kept

'' Data files are setup to have these boundaries with each line taking

'' exactly 64 bytes

wait\_d1 ' Set up data points to start of data storage location

OR OUTA,DI\_P ' Set input to known state

WAITCNT TARGET11,DELAY11 ' Delay for SD Card

MOV PTR11\_P1,#SPI\_DATA ' The state of local data store

MOV STOR11\_3,#32 ' 32 bits for each long word

MOV STOR11\_4,#64 ' Only clock in 64 long word

' wait for last bit of data token

' to drop low

wait\_dd WAITCNT TARGET11,DELAY11 ' Generate clock pulses until

OR OUTA,CLK\_P ' we see data

WAITCNT TARGET11,DELAY11

WAITCNT TARGET11,DELAY11

ANDN OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

MOV STOR11\_2,INA

AND STOR11\_2,DO\_P wz

if\_z JMP #spi\_l\_d1

JMP #wait\_dd

spi\_l\_d1 '' Clock in bits one at a time

'' If we are at the offset to offset+4

'' record the bits in data registers

'' Clock in next bit

WAITCNT TARGET11,DELAY11 ' Generate clock

OR OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

WAITCNT TARGET11,DELAY11

ANDN OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

' If we are at the offset start shifting in bits

MOV SPI\_OFFSET,SPI\_OFFSET wz ' Skip bits based on offset

if\_nz JMP #wait\_off

'' Look to see if we've read in all bits

'' if we have skip until 512 byte

'' have gone by

MOV STOR11\_3,STOR11\_3 wz ' We're now looking at one long

if\_z JMP #wait\_data ' word at a time, see if we're finished

' this word

'' Make the shifted register point to the correct

'' memory location, read in the bit and shift it

'' in to the data register

MOVD wait\_sm,PTR11\_P1 ' Write the new bit into local

MOV STOR11\_2,INA ' memory

AND STOR11\_2,DO\_P wz,wc

wait\_sm RCL 0,#1

SUB STOR11\_3,#1 wz ' Decrement the bit counter

if\_nz JMP #wait\_d2 ' Jump if we're not doen the word

ADD PTR11\_P1,#1 ' Move onto next local storage location

SUB STOR11\_4,#1 wz ' Decrement the local long word counter

if\_z JMP #wait\_d2 ' Jump if we've got all the words

MOV STOR11\_3,#32 ' Reload the bit counter for the next

wait\_data JMP #wait\_d2 ' long word

wait\_off SUB SPI\_OFFSET,#1 ' Decrement the offset until we've reached our bit

wait\_d2 SUB CNT11,#1 wz ' Move to the next bit

if\_nz JMP #spi\_l\_d1 ' otherwise we're done

wait\_d1\_ret RET

'' Send out clocks with data high until data line drops

'' This routine grabs the response from the SD card

'' At present nothing is done with the response

'' Grabs the first 8 bits

wait\_r1 OR OUTA,DI\_P ' Raise the data line

WAITCNT TARGET11,DELAY11

wait\_rr WAITCNT TARGET11,DELAY11 ' Generate the clock pulse

OR OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

WAITCNT TARGET11,DELAY11

ANDN OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

MOV STOR11\_2,INA ' Look to see if line is being

AND STOR11\_2,DO\_P wz ' pulled low

if\_z JMP #spi\_r1

JMP #wait\_rr

spi\_r1

spi\_l\_r1 MOV STOR11\_2,INA ' Clock in the first 8 bits of data

AND STOR11\_2,DO\_P wz,wc

RCL STOR11\_3,#1 ' Shift in the bit to store

WAITCNT TARGET11,DELAY11 ' Generate clock pulse

OR OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

WAITCNT TARGET11,DELAY11

ANDN OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

SUB CNT11,#1 wz

if\_nz JMP #spi\_l\_r1

AND STOR11\_3,#$7F ' Mask the lowest byte

wait\_r1\_ret RET

'' Send data out

'' Bytewise command lloks like

'' CMD - ARG - ARG- ARG- ARG - CRC

spi\_pck

MOV TARGET11,CNT ' Set up the first delay for

ADD TARGET11,DELAY11 ' clocking out data

MOV STOR11\_1,SPI\_CMD ' Command is only 8 bits

ROL STOR11\_1,#24

MOV CNT11,#8

CALL #spi\_byte ' Send the command byte

MOV STOR11\_1,SPI\_ARG ' Send the arg

MOV CNT11,#32

CALL #spi\_byte

MOV STOR11\_1,SPI\_CRC ' Send the (unused) CRC

ROL STOR11\_1,#24

MOV CNT11,#8

CALL #spi\_byte

spi\_pck\_ret RET

'' Send spibyte

''

spi\_byte

spi\_l\_byte ANDN OUTA,DI\_P ' Set up the first bit by setting

ROL STOR11\_1,#1 wc ' the output to a known state

if\_c OR OUTA,DI\_P ' and send out the bit

WAITCNT TARGET11,DELAY11 ' Generate the clock

OR OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

WAITCNT TARGET11,DELAY11

ANDN OUTA,CLK\_P

WAITCNT TARGET11,DELAY11

SUB CNT11,#1 wz ' Send out all 8 bits

if\_nz JMP #spi\_l\_byte

spi\_byte\_ret RET

DI\_P long $00040000

DO\_P long $00010000

CLK\_P long $00020000

SPI\_CMD long $00000000

SPI\_ARG long $00000000

SPI\_CRC long $00000000

TARGET11 long $00000000

DELAY11 long $00000100

STOR11\_1 long $00000000

CNT11 long $00000000

STOR11\_2 long $00000000

STOR11\_3 long $00000000

STOR11\_4 long $00000000

STOR11\_5 long $00000000

STOR11\_6 long $00000000

STOR11\_7 long $00000000

STOR11\_8 long $00000000

STOR11\_9 long $00000000

OUT11\_F long $4040

SPI\_SEC long $200

SPI11\_GO long $00005D60

SPI11\_RD long $00005D84

ONES11 long $FFFFFFFF

CNT11\_1 long $00000000

SEC\_DIR long 305664

SPI\_OFFSET long $00

SPI\_FLAG1 long $00000800

SPI\_HALF long $00000000

LU\_OFFSET long $0

PTR11\_P1 long $0

PERM\_W3\_M long $00000000

COSET\_W3\_M long $0

PERM\_W3\_L long $00000000

COSET\_W3\_L long $0

PERM\_W2\_M long $00000000

COSET\_W2\_M long $0

PERM\_W2\_L long $00000000

COSET\_W2\_L long $0

PERM\_W1\_M long $00000000

COSET\_W1\_M long $0

PERM\_W1\_L long $00000000

COSET\_W1\_L long $0

COSET\_LIMIT long $0

SPI\_COSET long $0 '' 1 is the first coset files 2 is the second

'' 3 is the third and 4 is the fourth

SPI\_DATA res 64