

Transportin' Cars...

Michael Majors
Jamie Stori
May 22, 1992

The goal of this project was to accomplish a complex mechanical task completely autonomously using a minimum of hardware. The task of loading an automobile from an upper train to a lower train is a relatively difficult problem. This project continually loads two cars from one train to another without operator interference and uses each mechanical element for more than one task.

Table of Contents

	Page #
List of Figures.....	i
1. Introduction.....	1
2. Operation.....	4
3. Hardware.....	5
3.1. Board Layout.....	5
3.2. Mechanical Elements.....	5
3.2.1. Automobiles.....	6
3.2.2. Robot Arm.....	6
3.2.3. Ramp.....	8
3.3. Sensors.....	13
3.4. Microcomputer.....	16
3.4.1. Inputs/Outputs.....	16
3.4.2. PAL.....	18
3.5. Circuits.....	21
3.5.1. Switching Circuit.....	21
3.5.2. Track-Kill Circuit.....	22
3.5.3. Servo Power Supply.....	22
4. Software.....	24
4.1. MJ.ASM.....	24
4.1.1. Initialization.....	24
4.1.2. Main Sequencing.....	25
4.1.3. Subroutines.....	25
4.2. Lecky Version 3.2.....	27
5. Error Correction.....	29
6. Conclusion.....	31
Appendices	
A. Schematics.....	33
B. Flow Charts.....	38
C. Code.....	45

List of Figures

<u>Figure</u>	<u>Page #</u>
1. Board Layout.....	3
2. TIL 311 Displays.....	4
3. Robot Arm : Side View.....	6
4. Robot Arm : Top View.....	7
5. Ramp : Top View.....	9
6. Ramp : Side View.....	10
7. Alignment Walls.....	10
8. Ramp Cross-section.....	11
9. Pivotal Device.....	12
10. Rotational Governor.....	12
11. Frontal Support.....	13
12. Hall Sensor Mount.....	15
13. Edge Connector Pin-outs.....	17
14. PAL Pin-outs and ADL.....	19
15. PAL Worksheet.....	20

INTRODUCTION

1

The objective of this project was to design an N-gauge model train layout that would sense something, actuate something, and sequence events. The project was to be built around several mechanical and electrical constraints. It had to fit the standard 'test-stand', that is to say be confined to a 2'x4' board and must contain the North and South test-stand tracks as well as the bi-directional Center track (See Figure 1). Furthermore, the project must interface with a Hornby Zero-One™ controller using a 6502 microprocessor-driven computer. The computer must run the Lecky collision avoidance software and connect to the test stand through the signal sensor board.

The constraints imposed by the designers above and beyond those detailed above were for the project to operate completely autonomously and that if possible, problems should be approached with simple mechanical solutions rather than complex electronics. The scenario chosen was the task of unloading two small automobiles from an elevated train and loading them on to a lower train. This is a fairly difficult operation, however the solution chosen minimizes both sensors, circuitry and mechanical components by utilizing each part in more than one manner.

The board was laid out so that one train may enter the project on the lower level while another can be diverted on to an elevated loop (See Figure 1). Thus two trains are able to run simultaneously on the project. Each train tows three cars. The first car is always a 'positioning' car and contains a freight container. The second and third cars are standard box cars modified with an internal ramp. One train should have its cars loaded with an orange Porsche and a yellow Ferrari (See Set-Up instructions on project-board for more detailed directions on initial train placement).

The sequencing of the operation is very simple. Each sequence of events is identical to the one preceding it. The sequence is as follows. The South train enters the board and is routed to the elevated loop and it advances to the hall sensor, stops and waits for the other train. The North train enters the board and is routed to the lower loop. It too advances to the hall sensor and waits. Once both trains have reached the sensors, the loading sequence

begins. Both trains advance from the 'positioning car' to the first freight car. The robot arm lowers the ramp, rotates to the elevated train and lifts the automobile from the freight car with a permanent magnet. The arm then rotates back to the ramp which pushes the car from the magnet, causing it to roll down the ramp and into the waiting lower freight car. The arm then lifts the ramp. Once the ramp has been lifted, the trains advance to the second freight car and the loading sequence repeats.

Once this loading operation has taken place the trains begin their sequence to switch positions such that the loaded train, now in the lower loop, travels to the elevated loop, and the empty train, now in the elevated loop, travels to the lower loop. This is accomplished through the following sequence. The upper track-kill is released and the train leaves the test stand. As it passes the North Block Clear, the South Block Track-Kill is over-ridden and the train advances to this position and waits. Now the lower track-kill is cleared and the engine leaves the board. As it passes the North Block Clear, the over-ride is removed and the sequence entire sequence begins once more.

The project thus accomplishes all the stated goals of having continuous hands-off operation using only a minimum of hardware. Furthermore, it accomplishes a difficult task using only mechanical devices rather than a plethora of circuitry and sensors.

TRACK LAYOUT

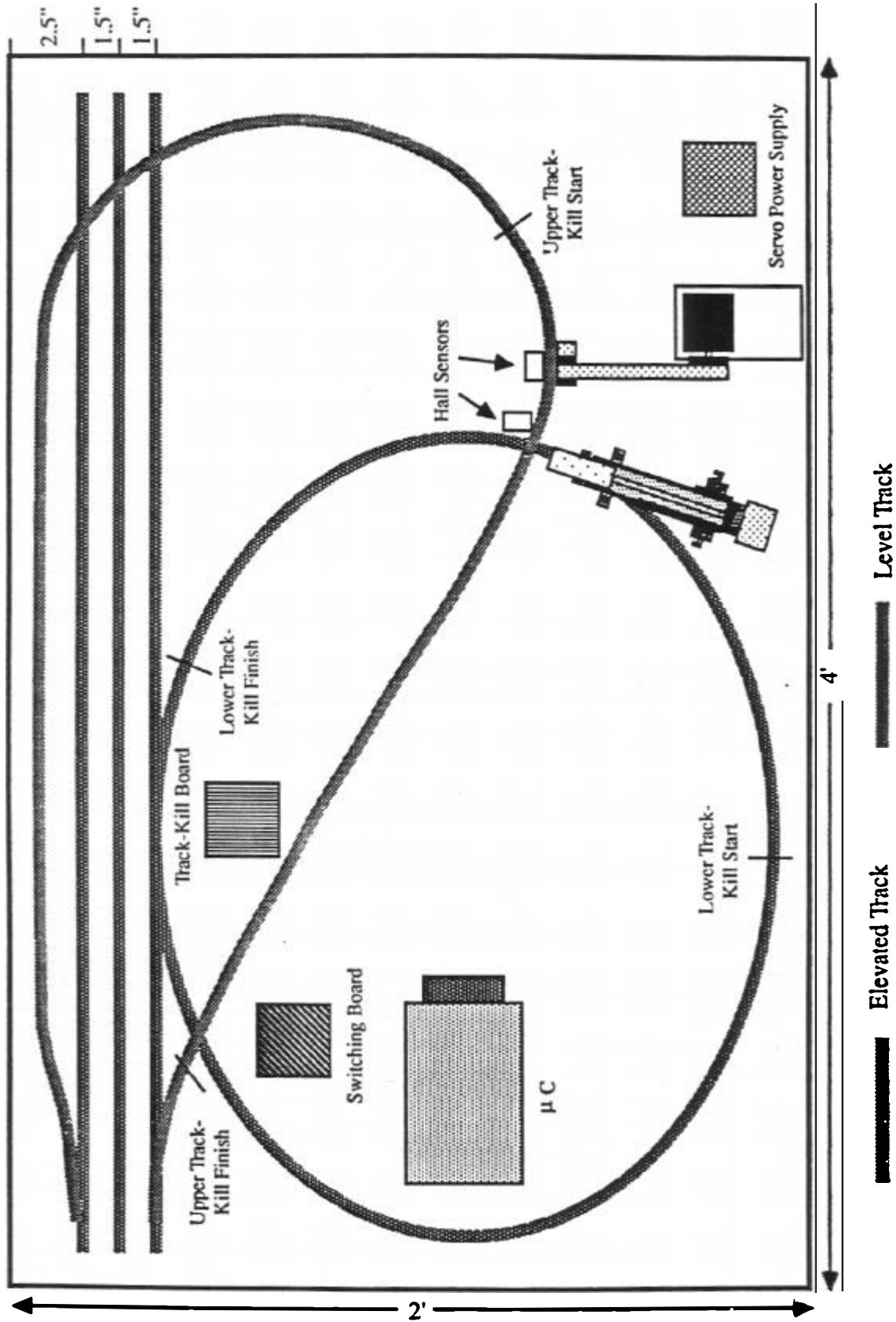


Figure 1 : Board Layout

OPERATION

2

Due to the design of the project, the operation of the train is very simple. Following the instructions on the project-board, one turns the Hornby speed control to 0, powers the Hornby, powers the servos, powers the computer, then turns the Hornby speed to the indicator. The project is designed to operate over a wide range of train velocities, however, if the trains are moving too fast the alignment of the loading positions will be adversely affected, thus the indicator should be viewed as an upper bound on speed. The engines chosen to work with this project are of the average-to-slow variety and should the need to replace them arise, new ones of this sort are desirable. However due to the difficulty in evaluating engine speed, some trial and error may be necessary to redefine the speed indicator.

Following the initial set-up, the project will run completely autonomously. The scenario has been designed such that it is self-perpetuating and it will thus run indefinitely. The computer displays shown below are simply to clarify the sequencing for the user and for diagnostic purposes--they do not require any action from the operator.

Figure 2 : TIL 311 Displays and Corresponding Train Operations

DISPLAY	OPERATION
11	PROGRAM INITIALIZATION
22	INITIALIZATION COMPLETE
AA	WAITING FOR TRAIN 1 TO REACH LOADING ZONE
BB	WAITING FOR TRAIN 2 TO REACH LOADING ZONE
10→1	COUNTDOWN FOR TRAIN TO COME TO FULL STOP
G0	ROBOT ARM LOADING AUTOMOBILES
1C	WAITING FOR N TRAIN TO CLEAR SENSOR
2C	WAITING FOR S TRAIN TO CLEAR SENSOR

HARDWARE

3

3.1 BOARD LAYOUT

The initial concept called for a transfer of materials from an upper to a lower train at a point of crossing of the two tracks. Figure 1 shows the final layout of the board. Rather than send the loaded train on an elevated loop, initially the climb and descent were to occur on a simple diversion of the North track. The grade required for such a feat surpassed project restrictions, and this concept was dismissed. In order to raise and lower the train to the height required for crossing the other track, a loop was necessary. Due to the size constrictions of the project, the best compromise was found to be a loop that broke off from the South track immediately upon entering our board, climbed steadily until high enough to cross over the tracks, and looped back to reenter the main lines close to where the North track leaves the board.

This caused some conflicts with the test stand's block collision avoidance system. As a result of the elevated loop, the loaded train never passes the South track clear sensor and the North track bar code reader. To allow for the desired movement of the trains, it was necessary to externally trigger the South track clear sensor, and make use of the North block override available in Lecky V 3.1.

Hall sensors are placed on the upper and lower loops to stop the trains at the loading and unloading positions. Isolated track sections are present at both of these areas to work in tandem with the Hall sensors.

3.2 MECHANICAL ELEMENTS

The project employed three mechanical and electro-mechanical devices. These were the 'nuts and bolts' of the operation. They provide all the action on the set-up as well as the error correction.

3.2.1 AUTOMOBILES

The automobiles transported from train to train in this project were standard, off-the-shelf MicroMachines™. They required little modification. Unfortunately, the difference in size between the Ferrari and the Porsche caused much grief. To standardize the car size, they were both filed. This also had the added advantage of giving them a more uniform shape. In order for the robot arm to be able to lift the cars magnetically, a small rectangular piece of stainless steel was glued horizontally to the roof of each car. With these two changes, any small toy could easily be utilized in this project. However there is one caveat; both the front and back of the vehicle must be fairly flat (i.e. no Batmobiles) for the detachment procedure to function properly.

3.2.2 ROBOT ARM

The robot arm was designed as a two degree of freedom system. This was accomplished through the use of two servos. The lower servo providing rotational motion and the upper servo on giving vertical freedom. Control of the servo motion is detailed in the software section. The plastic arm was glued to the upper servo and a permanent magnet was glued to the end of the arm (See Figure 3 Below).

Robot Arm : Side View

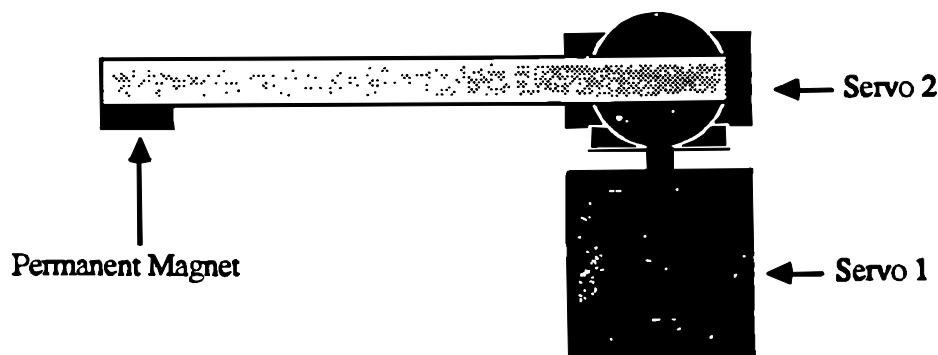


Figure 3 : Robot Arm : Side View

Attached to the magnet is another piece of plastic which facilitates detachment of the cars from the arm during unloading (See Figure 4 Below).

Robot Arm : Top View

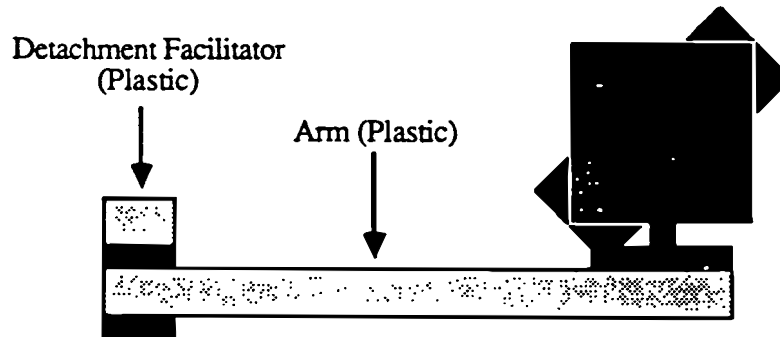


Figure 4 : Robot Arm : Top View

The robot arm has double duty. It loads and unloads the cars from the trains and also raises and lowers the ramp. The first part of the arm's motion is concerned with lowering the ramp. The arm's natural position is holding the ramp up. This is accomplished by the arm pressing down upon the back part of the ramp until it pivots up and is stopped by its rotational governor (See Section 3.2.2 Below). Beginning its motion, the arm lifts up and turns until it is aligned above the automobile to be unloaded. It then moves downward until it is approximately 2 millimeters above the automobile. The magnetic field of the permanent magnet chosen is strong enough so that the automobile is lifted without any possibility of the arm knocking the train car. Once the automobile is attached to the arm, it lifts and rotates to the ramp where the car is detached and it rolls into the waiting train-car.

The detachment of the car is accomplished simply. Using a permanent magnet simplified the design of the robot arm considerably, yet made the dropping of the car slightly more complex. This problem was solved by having the arm rotate the automobile's rear into a block at the back of the ramp and continue to rotate, while the automobile could not, until it dropped into the ramp. However, one unforeseen difficulty cropped up--the fringing effects of the permanent magnet caused the cars to slide off the bottom of the magnet and to swing around and attach to the front of the magnet. This was also easily mechanically solved. A small piece of plastic the size of the magnet (the 'detachment facilitator') was

attached to the side of the magnet where the cars would attach themselves. Now, instead of swinging upward, the cars would simply drop into the ramp properly.

The final function of the arm was to lift the ramp. This was accomplished, as stated above, by continuing the rotational motion used to detach the car until aligned with the back 'lifting plate' of the ramp. Then the arm moves downward until the ramp is fully in the up position. The arm holds its position until called upon to load automobiles again.

3.2.3 RAMP

The ramp is the most crucial component of the project. Due to the precise fit of the automobiles in the train freight car, the automobiles must be loaded exactly straight or they do not fall into the freight car. The features detailed below (Figure 5) all contribute to the alignment of the automobile.

Ramp : Top View

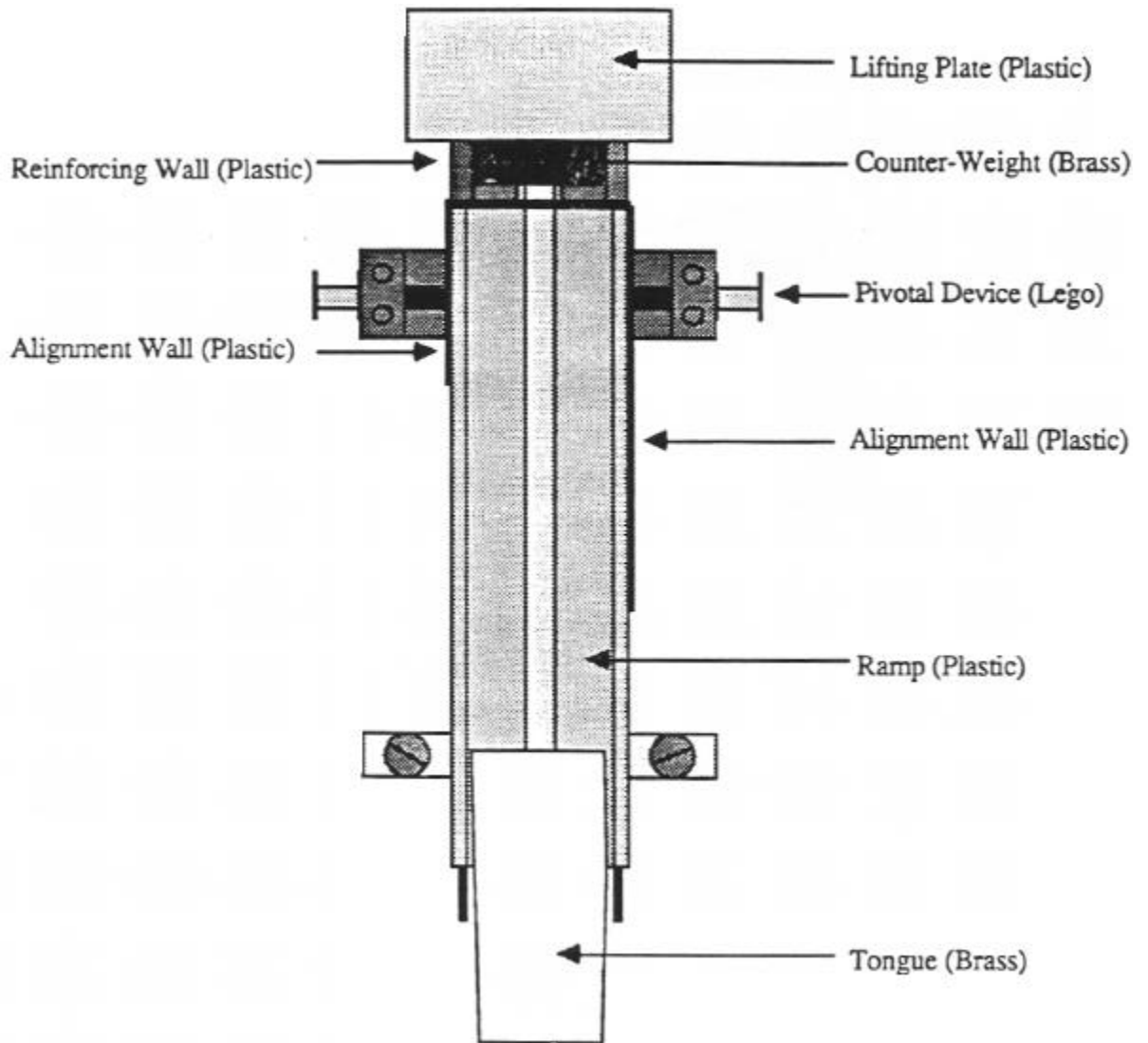


Figure 5 : Ramp : Top View

The various features of the ramp can also be seen below in Figure 6.

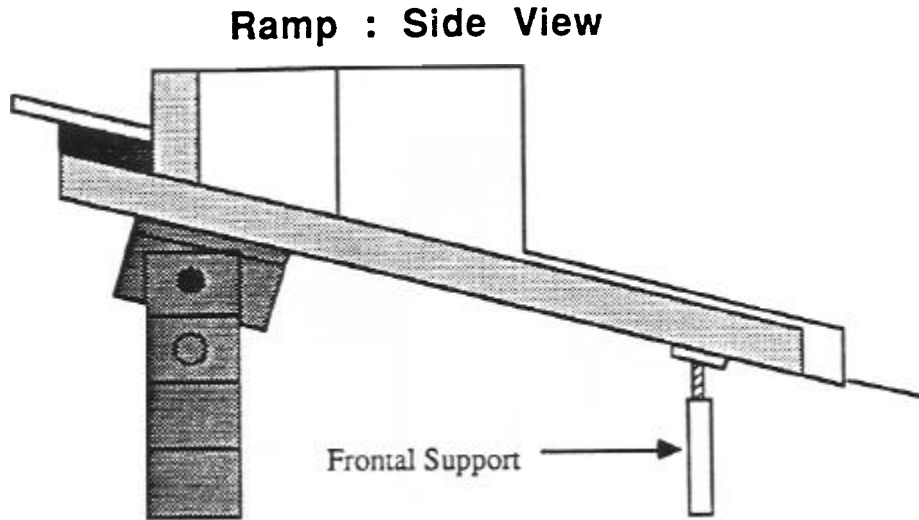


Figure 6 : Ramp : Side View

The first task of the ramp is to align the automobile on the robot arm so that it will drop properly into the ramp. This is accomplished through the use of the alignment walls seen below in Figure 7.

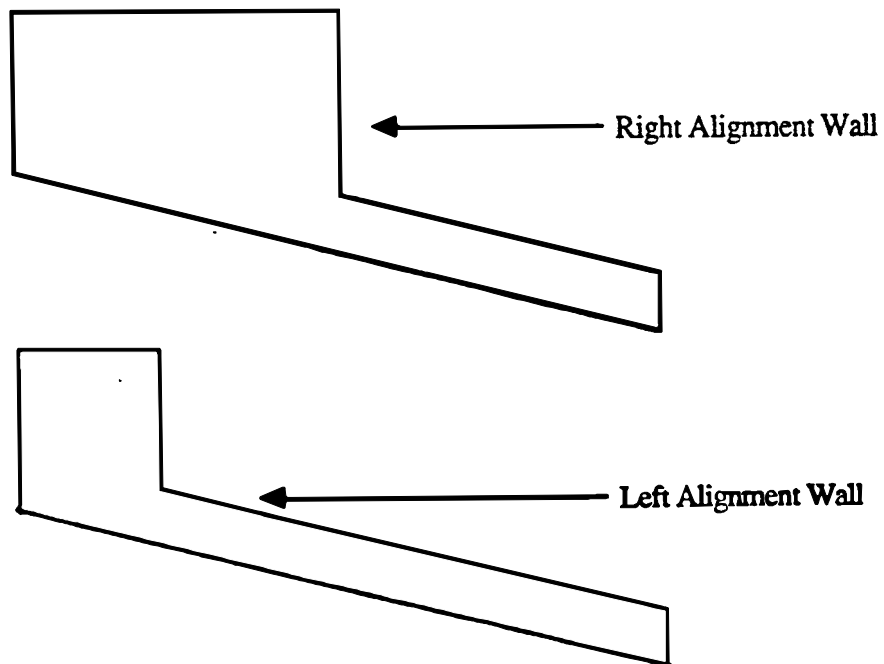


Figure 7 : Alignment Walls

It is possible that sometime during the arm's motion the automobile may rotate on the magnet. This would be detrimental to the detachment procedure since the automobile would fall on to the ramp at an angle and would not slide or load. Thus the right alignment wall is positioned such that a twisted car will be straightened as the arm rotates toward the back of the ramp. The left wall is cut further back so that the twisted automobile will have space to straighten during the arm's motion. As the arm nears the back of the ramp, the automobile is aligned precisely by both the left and the right wall. During the detachment procedure, the car is pushed off the front of the magnet and the two walls cause it to fall into the ramp properly aligned.

To keep the car from cocking once it has fallen into the ramp, the cross-section was designed to allow very little space between the sides of the car and the edges of the ramp. The automobile never cocks, and rarely, in situations of high humidity binds. The cross-section of the ramp-proper can be seen below in Figure 8.

Ramp Cross-Section



Figure 8 : Ramp Cross-Section

For the automobile to roll smoothly from the ramp into the waiting freight car, the ramp must be positioned just above the car. However, this does not allow the train to pass below. Thus, the ramp was built to pivot. As detailed above, the arm can raise and lower the ramp with a simple motion. But, the torques on the arm became quite extreme in this situation, so the ramp was counter-balanced to alleviate some of the forces. The pivotal device shown below in Figure 9 was constructed of Legos™ to provide a measure of modularity and for the resistance free rotation that they provide.

Pivotal Device

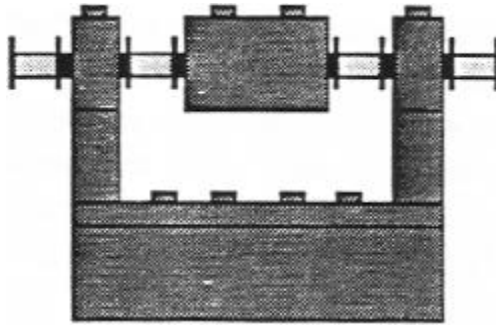


Figure 9 : Pivotal Device

Though the robot arm is relatively free from extraneous motion, occasional power-spikes lead to the arm lifting the ramp past its equilibrium position so that it became stuck in the up position. This problem was also conquered mechanically through the use of a rotational governor (shown below in Figure 10). This device, constructed of Legos, when positioned properly beneath the back end of the ramp, makes it impossible for the ramp to move past its equilibrium position. Thus, even if the arm surges, the ramp will never get cocked.

Rotational Governor

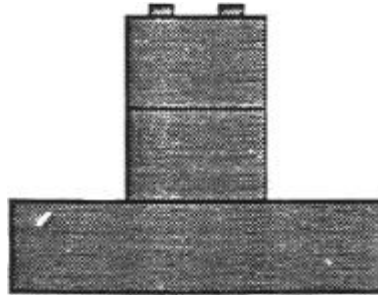


Figure 10 : Rotational Governor

The final obstacle to perfect loading of the automobiles was the side-to-side alignment of the ramp so the the automobile enters the freight car straight every time. Due to the constant up and down motion of the ramp, it was impossible to construct a pivot that would not allow some side-to-side variation without another point of support. Thus an alignment jig (See figure 11 below) was positioned on the board so that as the ramp lowered, it would fall into the exact same place time and again.

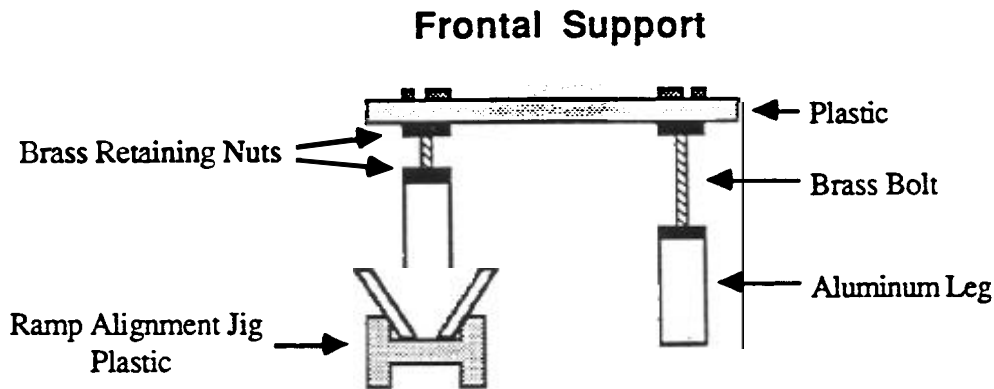


Figure 11 : Frontal Support

The frontal support was constructed with bolts and retaining nuts so that the height of the ramp can be adjusted to accommodate other freight vehicles if the user so desires. Both the bolts and the retaining nuts chosen were brass due to the non-binding and corrosion resistant nature of the material.

3.3 SENSORS

The primary goal of the project was to minimize all possible hardware. To this end, Hall Sensors were chosen over Optical Sensors. The Halls have built-in Shmitt Triggers and all they require is power and ground and they will do their thing. Thus, each freight car was equipped with a rare-earth magnet on the right side (to avoid the bar-code readers) for positioning the stopped train for the loading and unloading of the automobiles.

Sensor positioning was absolutely critical to the success of this project. Due to the nature of the objects being transferred from train to train, the alignment of the trains was at least as critical as the alignment of the ramp. An easy way to accomplish this is to slow the trains to a point where they stop as soon as their power is removed. However this requires either continually slow motion of the trains or user intervention. Since a primary goal of the project was to eliminate the need for the operator, and the slow trains could not make it up the inclined track, a new solution was needed. Thus it was determined that a positioning car would be used which would trigger the track kills as it went by. Placing this car as the first car in the train would allow the engine some time to coast to a stop as the positioning car passed the sensor. The computer would then clear the track-kill, allowing the engine to begin motion, however the short distance between the train's current position and when the next freight car triggered the hall sensor did not allow the engine to build any speed and the positioning of the car was very near to perfect. Furthermore, this set-up allows the addition of as many cars as the user wishes since the scenario is the same once the engine has come to a full stop after the positioning car.

The standard Hall Sensor mounting procedure in styrofoam is rather inelegant. Thus a new, and simple method was devised. Cutting a rectangular chunk of plexiglass and drilling two holes in it provides a neat and good-looking way of mounting the Hall Sensor as seen below in Figure 12.

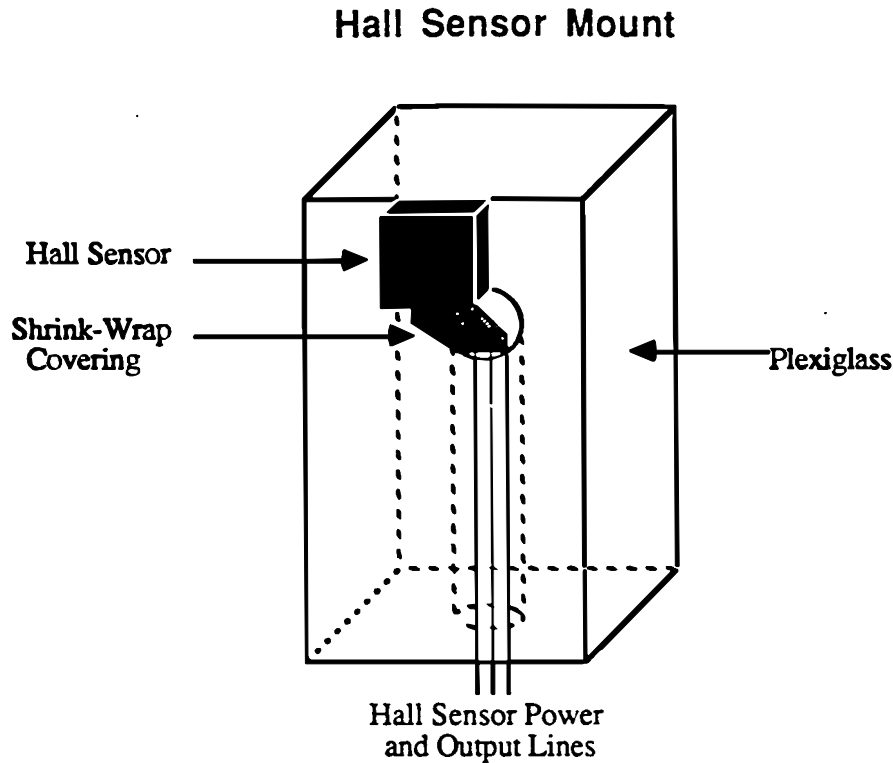


Figure 12 : Hall Sensor Mount

As per the primary goal of this project, minimizing all possible unnecessary hardware, the Optical Sensors on the Test-Stand were utilized rather than installing additional Hall Sensors on the project-board. The Lecky program and the test-stand provided two convenient sensors and track-kills. These, as detailed earlier, were fully utilized in the sequencing part of the project. However, to use the sensors two new wires needed to be added to the bus from the computer to the signal sensor board (SSB). These wires act as both inputs and outputs to the VIA's. This can only be accomplished through the use of open-collector logic. Conveniently, the designers of the SSB and the Lecky software set the test-stand up using open-collector logic. Thus it was possible to read the outputs of the sensors as well as false-trigger them to clear the track-kills. However, the use of the track-kills was also desirable. Thus, through the utilization of version 3.2 of the Lecky

software, it was possible to override the clear signal and use the track-kill as one normally would on the project board. The code for performing this operation is detailed in Section 4. Since the layout of the board necessitated the clearing of the South Track Clear, it was determined that even though the utilization of the sensors external to the board would not allow the board to be connected to other projects, this possibility had already been eliminated and the minimization of the number of sensors was a desirable quality of this set-up.

3.4 MICROCOMPUTER

The computer controlling this project differs only slightly from the standard version detailed in Class¹. The components unique to this computer are the buffering scheme and the Programmable Array Logic or PAL.

3.4.1 INPUTS/OUTPUTS

The inputs and outputs to and from the computer to the project-board and the signal sensor board are listed below together with their pin assignments on the 44-pin edge connector, the pin assignments on the computer, a description of the input/output function and the color code of the wire.

¹Class refers to MAE 412 taught by Professor M. G. Litman at Princeton University

EDGE PIN	COMPUTER	DESTINATION	COLOR
1	N/C		
2	N/C		
3	N/C		
4	GROUND	SSB DB25 PIN 15	BLACK
5	VIA A000 PB0	SSB DB25 PIN 16	WHITE
6	VIA A000 PB1	SSB DB25 PIN 17	YELLOW
7	VIA 8000 PA3	NORTH CLEAR DB25 PIN 7	BROWN
8	VIA A000 PB3	SSB DB25 PIN 2	ORANGE
9	VIA A000 PB6	SSB DB25 PIN 3	GREEN
10	VIA A000 PB2	SSB DB25 PIN 1	PURPLE
11	VIA A000 PB4	SSB DB25 PIN 4	RED
12	VIA 8000 PB5/PA2	SOUTH CLEAR DB25 PIN 8	BROWN
13	VIA 8000 PA1	INPUT FROM ELEV HALL	BLUE
14	VIA 8000 PA0	INPUT FROM LOOP HALL	BLUE
15	VIA A000 PB7	ROTATIONAL SERVO TIMER	WHITE
16	VIA 8000 PB7	VERTICAL SERVO TIMER	WHITE
17	VIA 8000 PB4	LOOP TRACK-KILL CLEAR	PURPLE
18	VIA 8000 PB3	ELEV TRACK-KILL CLEAR	PURPLE
19	VIA 8000 PB2	ELEV SWITCH THROW	PURPLE
20	VIA 8000 PB1	LOOP SWITCH THROW	PURPLE
21	VIA 8000 PB0	SWITCH DIRECTION	PURPLE
22	POWER	DAUGHTER BOARDS	RED
A	4N33 PIN 1	TRACK POWER	YELLOW
B	4N33 PIN 2	TRACK GROUND	GREEN
C-V	GROUND	DAUGHTER BOARDS	BLACK

Figure 13 : Edge Connector Pin Assignments

Each input and output listed above is routed through an Octal Buffer (74LS244) as it enters or leaves the computer for signal amplification and computer isolation respectively. The wiring scheme is detailed in Appendix A, at rear.

3.4.2 PROGRAMMABLE ARRAY LOGIC

The Programmable Array Logic or PAL is used to incorporate the microprocessor's address decode logic into a single chip. By burning selected fuses on the PAL, the 16-input AND-Gates can be configured to NOT-Gates, 2-input NAND-Gates and even 3-input NAND-Gates. The worksheet showing the selected fuses is shown on the follow page in Figure 15. Thus, the number of ICs on the computer is minimized and valuable 'real-estate' on the Scotch-Flex™ board is saved. The address decode logic for each chip on the computer and the pin-outs for the PAL are shown below in Figure 14.

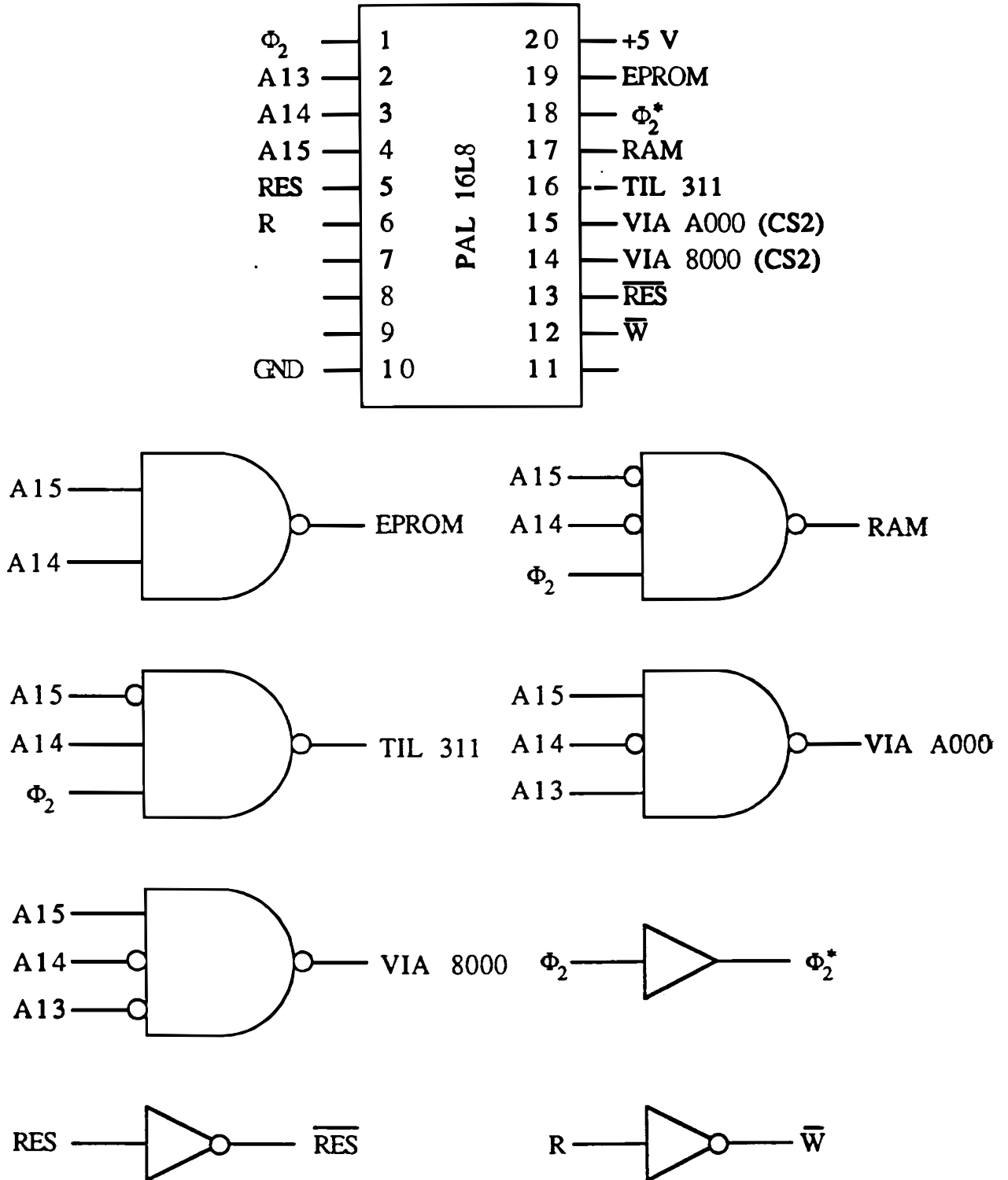
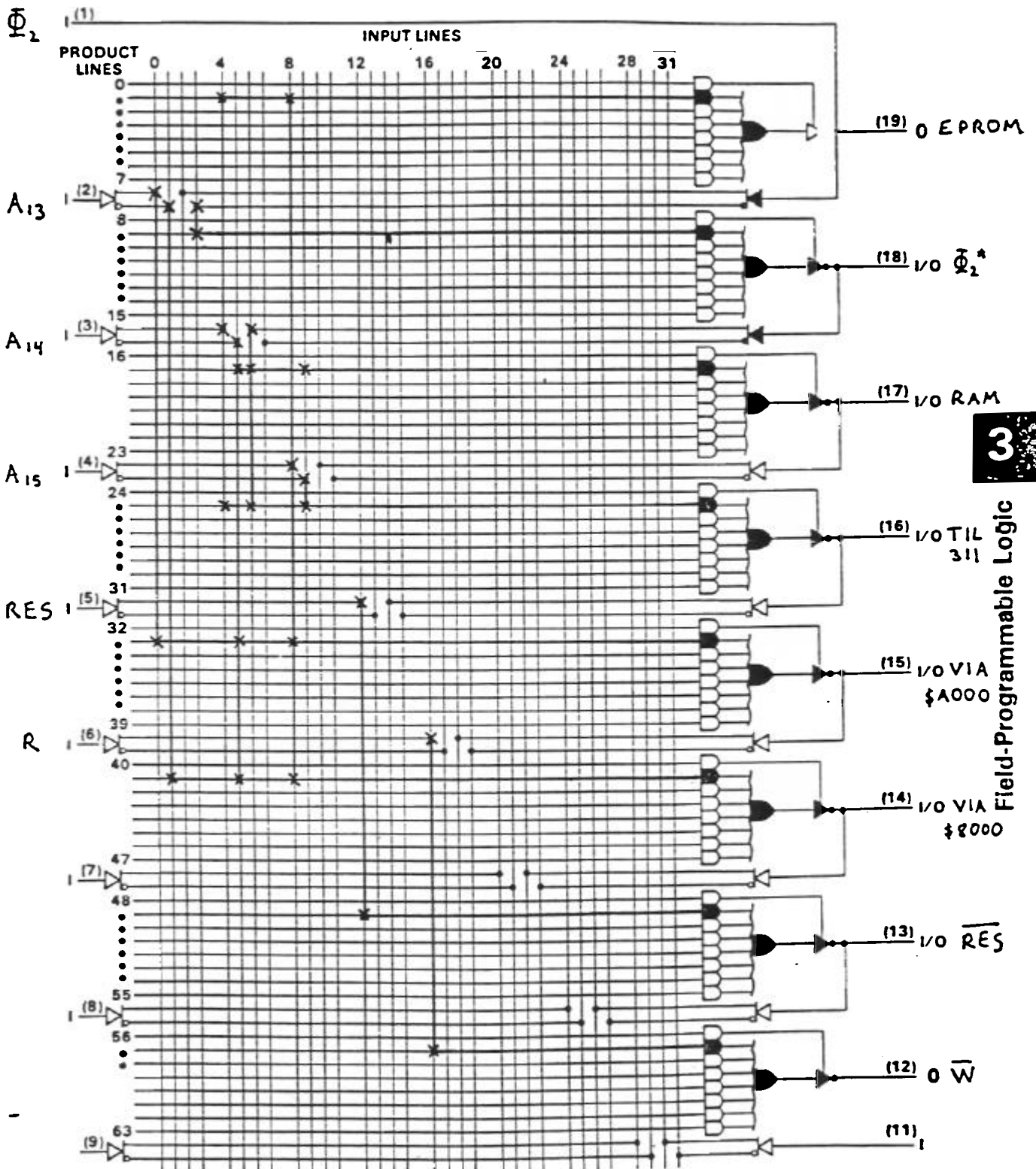


Figure 14 : PAL Pinout and Address Decode Logic



3.5 CIRCUITS

All circuits were designed with the intent of minimizing hardware while providing robustness of operation. In this spirit, the switching daughterboard controls four switches with only two SCR's and a single relay. The track kill circuits provide fail safe removal of power to the track sections through the use of Latching Flip-Flops which can then be externally cleared. The servo power supply proved necessary when all other attempts to smooth out the servo operation failed. The schematics of the circuits detailed below can be found in Appendix A.

3.5.1 SWITCHING CIRCUIT

Upon examining the sequencing, it was discovered that only four different switch positions were needed i.e. two pairs of switches rather than four individual switches. Consequently, there are only two switching circuits sharing a relay for directional control. A large source of trouble in many switching circuits is the double throwing of switches. An SCR is typically used for throwing the switch in each direction, and the sensitive nature of the SCR leaves open the possibility that the triggering of one might in turn trigger the other. To avoid this problem and minimize the circuitry, a double throw relay was used, with its coil controlled by an additional signal line to determine the direction of the switch throw. Because the relay was of the double pole double throw variety, the same relay could be used in the directional control for both pairs of switches.

The relay coil was thrown by a PNP transistor triggered by the TTL signal. A diode and capacitor were placed in parallel with the relay coil to eliminate ill effects of back EMF when power is removed from the coil. The actual switch throws were achieved with the standard Opto-isolator / SCR based throwing circuit. The TTL input signal is isolated from the Hornby power with a 4N33 Opto-Isolator. The output from the 4N33 triggers the gate on the SCR, and the reversing polarity of the Hornby power is relied upon to turn off the SCR.

3.5.2 TRACK-KILL CIRCUIT

There are two identical track-kill circuits on the track-kill daughter board. Because the sequencing called for cutting the power to the track immediately after the Hall sensor was triggered, a fail safe circuit was designed in order to relieve the software of this task. The Hall sensor outputs ground when a magnetic field of 70 gauss is present, and is high otherwise. This signal was inverted and used as the clock input on a D Type Latching Flip-Flop with preset and clear. (74LS74) The D input was held high, so that when clocked, the the output became high. The clock triggered on a rising edge, which corresponded to a falling edge of the Hall sensor.

To clear the Flip-Flop, a computer signal ran directly into $\overline{\text{CLR}}$ on the 74ZLS74. The output from the flip-flop was then used to drive a relay coil exactly as described above in the switching circuit. The track power to the kill section of track was run through the relay so that the connection was broken on the throwing of the relay.

An unexpected problem with this circuit was an undesired triggering of the flip-flop caused by electrical noise on other parts of the board. Most commonly, throwing a pair of switches would occasionally trigger the track kill circuit. The remedy employed was to tie the sensor inputs high through a fast acting capacitor. (0.1 mF) The capacitor buffered the signal enough to smooth out small surges caused by other devices.

3.5.3 SERVO POWER SUPPLY

The Servo motors proved to be extremely sensitive to noise in their power supply. They were initially run from the computer's power supply, but this provided unsatisfactory performance. Attempts to smooth out the power supply with despiking and decoupling capacitors did not provide enough isolation for smooth operation of the motors.

The solution implemented was an entirely separate power supply built solely to power the servos. A 9 V DC power input was run through a 7805 voltage regulator which provided the desired 5 volts. An 100 mF decoupling capacitor was run between the input to the 7805 and ground. Five 0.1 mF despiking capacitors buffered the 5 V output..

Opto-isolators were used to bring the computer servo signals to the reference voltage of the new power supply. The computer signals and grounds triggered the LED's of the 4N33's. The base of the photo-transistor was grounded through a 470K resistor to provide the required fast response from the 4N33. The output from the photo-transistor was run through a 1K resistor in order to bring it down to a TTL level signal.

SOFTWARE

4

The Assembly Language software for the 6502 based microcomputer was developed on an IBM PC/AT using the KEDIT text editor. Once written, the programs were compiled using the XASM65 cross-assembler by Avocet. The compiled code was then burned onto an EPROM using a GTEK Eprom programmer and supporting software.

4.1 MJ.ASM

The Assembly Language code of the final software, MJFINAL.ASM, can be found in Appendix B at the end of the report. As explained earlier, the project is not interactive - it was designed to perpetually and autonomously perform its task of transporting the automobiles back and forth between two trains. Subsequently, the main flow of the program is a repeating sequence of events. For flowcharts detailing the program flow, see Appendix A. What follows is a verbal description of the program's operation that attempts to fill holes in the detail given by the flow charts.

4.1.1 INITIALIZATION

The first task is initialization of the VIA's. Port B of the VIA at address \$8000 is set to be an output port, and port A is set up as an input port. The simplest way to achieve simultaneous control of the two servos was to make use of Timer 1 in one-shot mode on both the \$8000 and \$A000 VIA's. After a very slight modification to the Lecky code (see following explanation, section 4.2) we were able to use both Timer 1's without problem.

After initialization of the VIA's, the arm was moved to its wait position, that of elevating the ramp. At this point, a screen display of AA indicates that the user can start up the trains, and the sequencing begins.

4.1.2 MAIN SEQUENCING

Switches are thrown into the ELEV and INLP positions, and both track kills are cleared. The HWAIT procedure is then called, which waits for both trains to arrive at the Hall sensors. At this point the track kills are again cleared, and once again HWAIT is called. At this point, the first car is now aligned for loading. ARMG sequences the motion of the arm. The one difference between the first and second car transfer is a slight difference in the position attained by the train on the upper track. To adjust for this, the low order byte of the stop position is stored in STL B, and ARMG moves to that position.

After completion of one car transfer, both track kills are cleared, HWAIT is called, and the trains advance to the next car. At this point ARMG is called again with a new value in STL B. The task of the software is now to switch the position of the two trains. To do this using a minimal number of sensors, we externally taped into the North and South Block clear sensors. Additionally, our elevated loop of track circumvents the South track clear sensor and the North track bar code. This required external clearing of the South block clear sensor. After clearing the South Block, the software clears the upper track kill, and waits for the empty train to clear the North track clear sensor.

The THGH switch position is now thrown, and the North Block Override is set. This was necessary because the upper train never passes the North Track bar code reader. At this point, the program waits for the train to clear the South Track clear sensor. Now the lower train can be released after the switches are set to the OUTL position. When the lower train clears the North Block clear sensor, the North Block override can be released, and the sequencing can now be repeated.

4.1.3 SUBROUTINES

The switching procedures are fairly straight-forward. Two of the switch positions, INLP and THGH require throwing the relay prior to triggering the SCR. Additionally, the relay must be held in this position for two frame cycles from the time of the SCR triggering. The SCR is not turned off until the AC current changes polarity, and since the polarity flips on alternate frames, waiting two frames is the only way to assure reliable operation. As

per the flowchart, the switching procedures all wait for FRANUM=1 before starting their operation. This assures that all action can be completed before the data frame begins, as per the project requirements. The Signal Clearing procedures follow the same format. To avoid conflicts with the data frames, the signal is held low for all of FRANUM=1.

A good deal of the operation of our project involves waiting for trains to attain their position. In addition to waiting, however, the software must continue to send regularly spaced pulses to the servos in order for the arm to hold the ramp in its elevated position. What we felt to be the simplest solution to this problem is implemented in the procedure SPUL. The SPUL procedure is used to send pulses to the servos both at the start of frame one and in the middle of frame three. Because a full cycle is really five frames long and our software is not active during frame five this appeared to be the best timing option. After sending a pulse, the current frame number is stored in LSTP. The next time SPUL is called, it will only output a pulse if FRANUM is not equal to LSTP. Then, if it is in frame one, a pulse is sent out immediately, while if it is in frame three, there is a delay for roughly half of the time of the frame. If SPUL is called often enough (i.e. every time through a loop of reasonable length,) the resulting output to the servos will be a fairly regular stream of pulses.

A straightforward use of the SPUL procedure is made by the NBKR and SBKR waiting procedures. These procedures simply loop until the desired block clear signal goes low. Each cycle of the loop, SPUL is called, in order that the arm hold the ramp up while the trains switch positions.

The HWAIT procedure loops on the Hall sensor outputs until both Halls have been triggered. SPUL is also called during each cycle of the loop. While waiting for the first train to arrive, all but bits 0 and 1 of input port A are masked off. The procedure then loops until the result does not equal three, i.e. one of the Halls has gone low. The mask for the second train then isolates the input bit of the Hall which has not yet triggered. When this result is zero, both trains have reached the Hall sensors and triggered the track kills. The termination of the HWAIT procedure is a delay to allow the later arriving train to come to a complete stop. A roughly one second delay is created by calling FDEL with a value of 10. FDEL effectively waits then for 10 switches from FRANUM=1 to FRANUM=2.

The motion of the arm is controlled by the ARMG procedure which makes calls to the MV procedures for each movement. Control of the servo motors is achieved by sending out a pulse train at slightly less than 60 Hertz through the use of the SPUL procedure described above. The width of the individual pulses dictates the position that the servo will attain. These pulses are generated by the two one-shot timers on the VIA's. The width of the pulse corresponds to a two byte number which must be loaded into the timer addresses.

The high and low order bytes for the pulse lengths for the horizontal and vertical motion servos are stored in HTLB, HTHB, VTLB, and VTHB. Moving the arm requires no more than slowly changing the pulse width while sending out pulses to the servos. The simple procedures UP, DOWN, RGHT, and LEFT perform either a two-byte addition or subtraction on either the horizontal or vertical pulse width data variables. The MV procedures do no more than alternate calls to a subtraction or addition with calls to PULS until a stopping position is obtained. The stopping position must be stored in SPLB, SPHB prior to the call of the MV procedure.

The PULS procedure is a simple extension of SPUL. PULS stores LSTP in a temporary variable TLSTP and loops on SPUL until LSTP and TLSP pulse are no longer the same, in other words, until a pulse is sent to the servos. Controlling the speed of the movement then is reduced to varying the increment of the addition or subtraction done each pulse cycle.

4.2 LECKY VERSION 3.2

Historically, both timers on the \$A000 VIA have been reserved for the Lecky background software. Upon inspection of the Lecky code, it was discovered that Timer 1 was not made use of at all. In the Lecky initialization routine, it was found that the PB7 output of timer one was disabled even though the Lecky routine did not make use of the timer. The sole modification that has been made to V3.1 is in the initialization of the ACR. Lecky V3.2 does not affect bits 6 and 7 of the ACR, the bits that determine the mode of timer 1. If one wishes to use timer 1 on both VIA's, using V3.2 is the cleanest way of doing so.

The unavoidable ill affect of using this timer is to squelch any other output to that pin. PB7 is reserved by the Lecky routine as an output indicating whether or not the Hornby is engaged in a data frame. As long as Timer 1 is in one-shot mode with output to PB7 enabled, there is no access to this information.

ERROR CORRECTION

5

The design of the project attempted to solve all errors mechanically rather than through the software. Due to the nature of the project, the possible errors were few. They are, de-railing of one or both engines, and misloading of one or both automobiles.

The de-railing of the engines/cars is the more difficult problem to solve. The computer is able to detect this problem easily. If the engine does not arrive (trigger a sensor) where it is supposed to be within a reasonable amount of time, the computer can time out, alert the operator and halt the flow of the program. However, this did not appear to be a very effective, or even useful sort of error correction. Since the computer is simply waiting for the engine to arrive at a sensor and no other actions are taking place simultaneously, halting the program only has a detrimental effect. The operator will, in all likelihood, be alerted to the de-railing of the engine before he/she is aware of the computer signalling. It is far more advantageous for the operator to simply re-rail the engine and allow the program to continue than to implement the error correction described earlier.

The second sort of error that is possible in this scenario is the misloading of the automobiles into the train's freight car. This error was solved mechanically. The first possible method to correct this problem is to design the system to be so robust that this error never occurs. Since it is virtually impossible to foresee all types of errors that may occur in a system as complex as the one chosen, designing for robustness is still desirable, but not the solution to every problem.

The errors observed in the automatic loading of the cars are these: the car is misaligned on the magnet when it is lifted by the robot arm; the car is loaded into the freight car crooked; the car does not detach or roll properly in the ramp. All these errors were corrected through the design of the mechanical components of the system. The first problem was the simplest to solve and was mentioned earlier in the design of the ramp. The alignment wall of the ramp are positioned such that as the robot arm executes a level arc moving toward the back of the ramp, the car will automatically be straightened on the magnet. The car will slide along the right wall until it reaches the section where the left wall begins. At this point the car will have made contact with the back wall and the detachment will begin with the car pointing directly down the ramp.

The second type of car-loading error is when the automobile rolls into the freight car improperly and it does not seat correctly. This error was initially addressed through the precise alignment of the ramp as detailed above. However, the problem still occasionally persists and is further mechanically corrected. When the automobile is tipped in the freight car, it appears that the robot arm will not be able to lift it properly. However, the permanent magnet was chosen such that the field was strong enough to lift the automobile without making physical contact. Thus, the 'tipped' car will be straightened on the arm and the loading sequence will continue as before. Thus the problem was solved through robust mechanical design rather than complicating the procedure with the use of more sensors.

The last type of error is the most difficult to solve and categorize. The automobiles occasionally 'stick' in the ramp due to climatic changes or other such unavoidable circumstances. This was solved through the use of a precisely dimensioned ramp and the choice of automobiles with very low rolling resistance.

Thus, the error correction in this project was not implemented in the traditional sense, but instead was built directly into the mechanical components. It was hoped that this sort of design would be more robust in the long run and less sensitive to problems with sensors and other such delicate electronic components.

CONCLUSION

6


The intent of the project was a perpetual transfer of the toy automobiles between two trains. To this end, a mechanical solution was implemented to accomplish the task in as straightforward a manner as possible. The ramp and robot arm are designed to transfer the cars back and forth between the trains despite small differences in position of the cars at both ends of the loading process. The alignment walls straighten out the cars until the back wall, with the aid of the detachment facilitator, knocks the car off the permanent magnet, and it rolls into the waiting train. The operation of the project is fairly robust, though not flawless. Slight misalignment of the ramp results in misloading of the cars. In addition, large speed discrepancies can result in large position displacements that will result in missed pickups by the arm.


This project resulted in a couple atypical solutions to problems. The avoidance of one end of the test stand by our elevated loop required an open collector triggering of the South track clear sensor, in addition to the use of the North block override available in Lecky V3.1. This allows for operation of our project on a test stand only. Working in tandem with other projects was not a consideration in our design, and would require modification to the software.

Use of Timer 1 on both VIA's is something new to this course. Although the Lecky routine does not make use of Timer 1 on the \$A000 VIA, the output to PB7 is used as an external indication of the presence of a data frame. Lecky V3.2 is the result of a slight modification to the code which allows use of Timer 1 by not resetting it in the initialization of the software. The result is an inability to access the typical PB7 output.

The project represents a somewhat unique solution to a fairly difficult task. The combination of two servos to produce a simple arm with two degrees of freedom allowed for a wide range of motion that could be easily and accurately controlled with little supporting hardware. There is much that could be done with these general concepts, and perhaps it will serve as an inspiration to future projects.

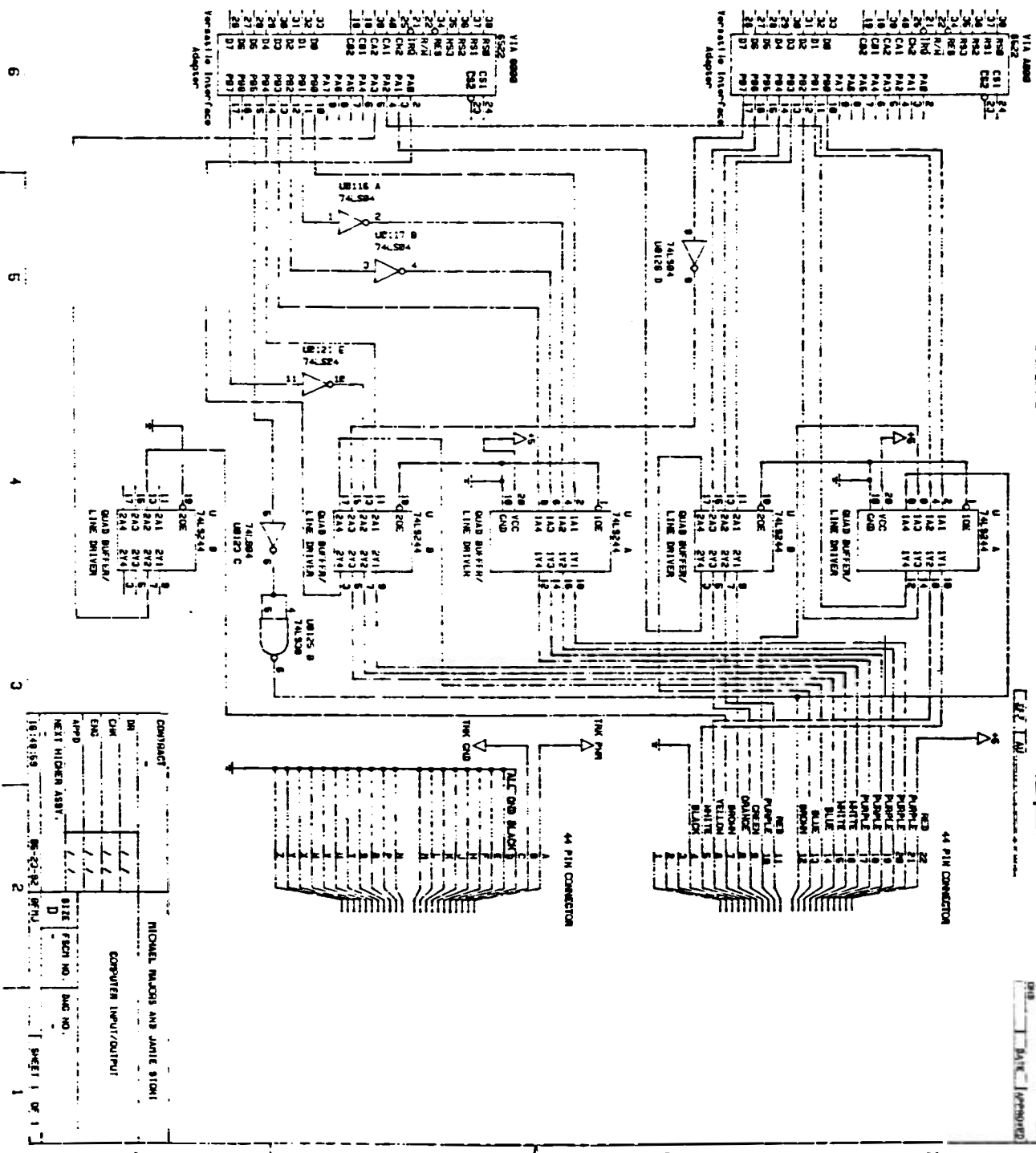
I pledge my honor that this paper has been completed in accordance with University Rules.


Michael D. Majors

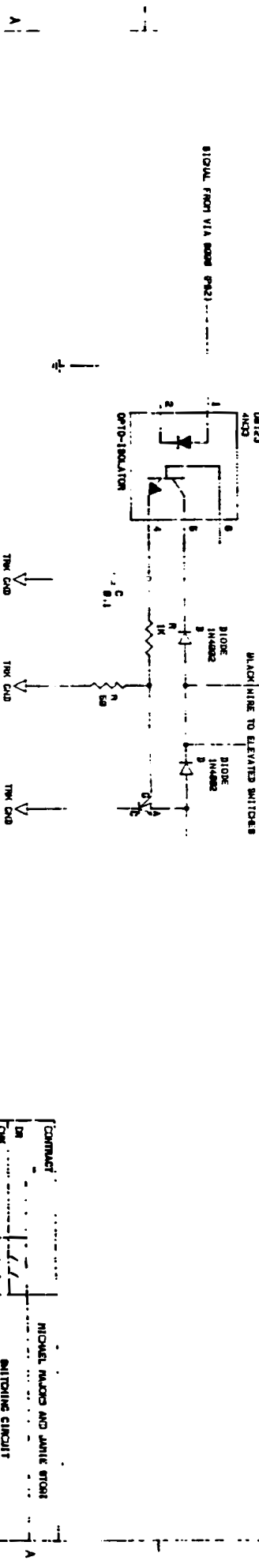
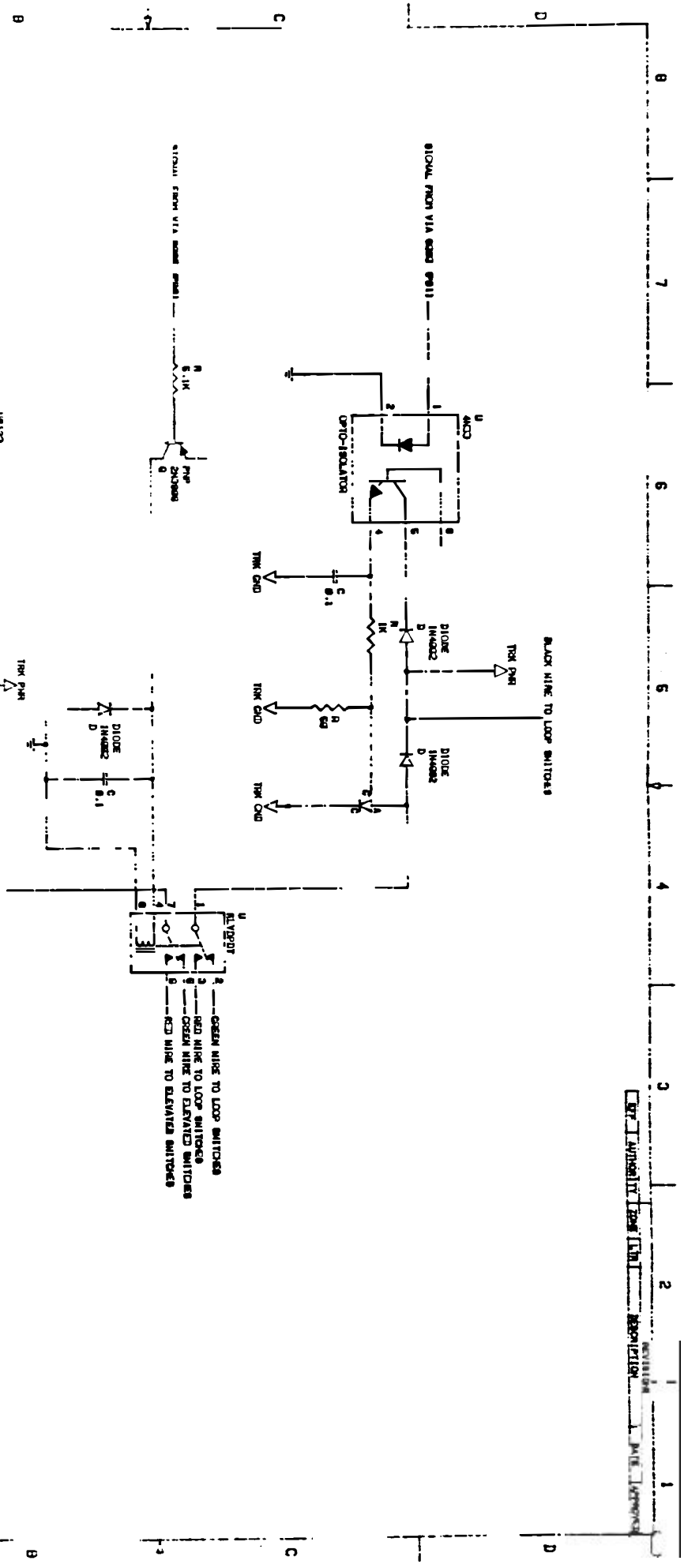

James A. Stori

APPENDIX A

SCHEMATICS



CONTRACT	MICHAEL, RAJON AND JANIE STOKI
CHK	COMPUTER INPUT/OUTPUT
END	
APPD	SIZE FECH NO.
NEXT HIDDEN ASSY.	D
18:08:39	19:02:42
1	1

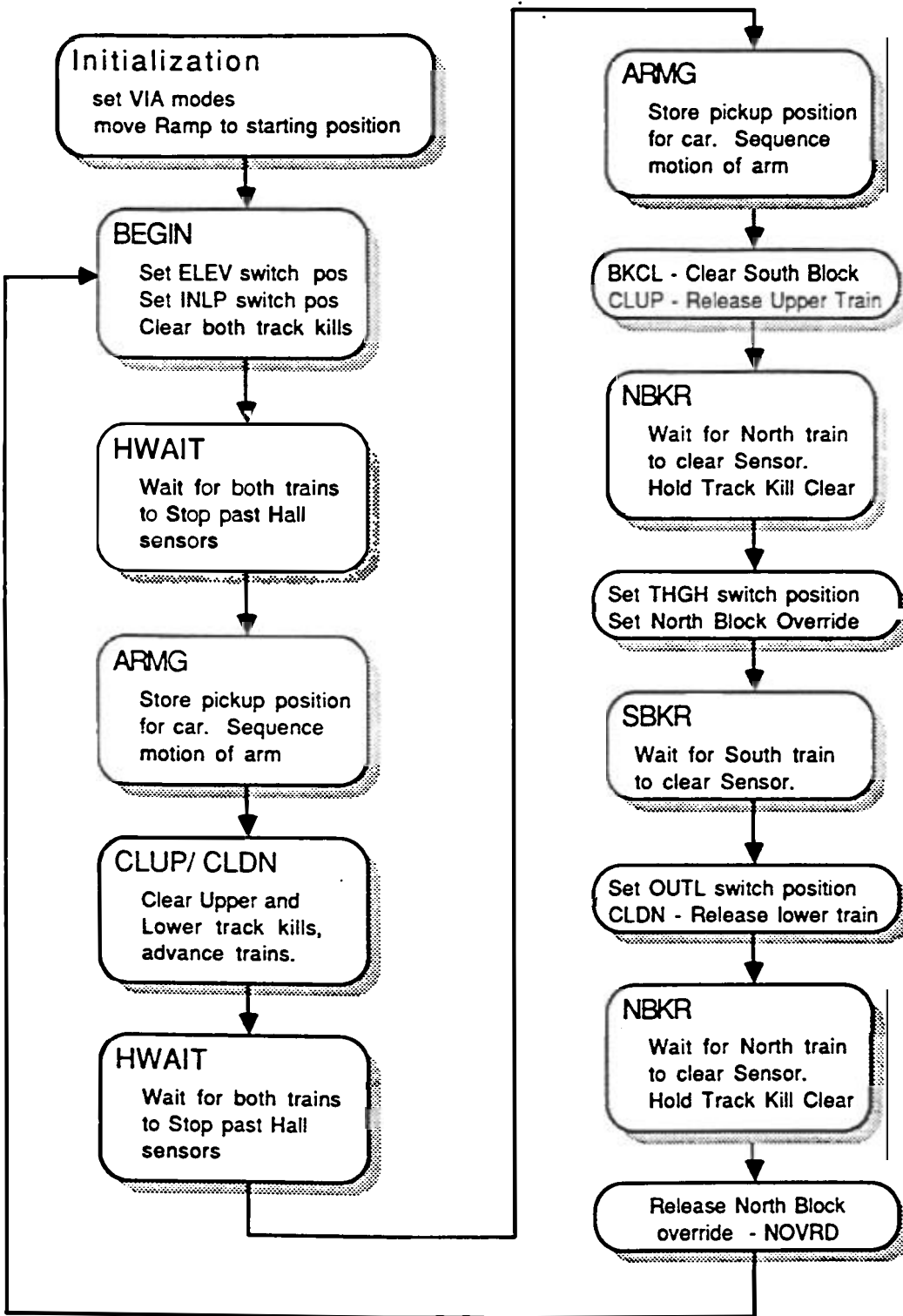


CONTRACT		MICHENER, RAJONS AND JARVIS STORE	
DR	100-100-100-100	DATE	1/1978
END	100-100-100-100	SIZE	D
APP'D	100-100-100-100	SHEET NO.	1 OF 1
NEXT MICHENER ANY		DATE	
100-100-100-100		100-100-100-100	

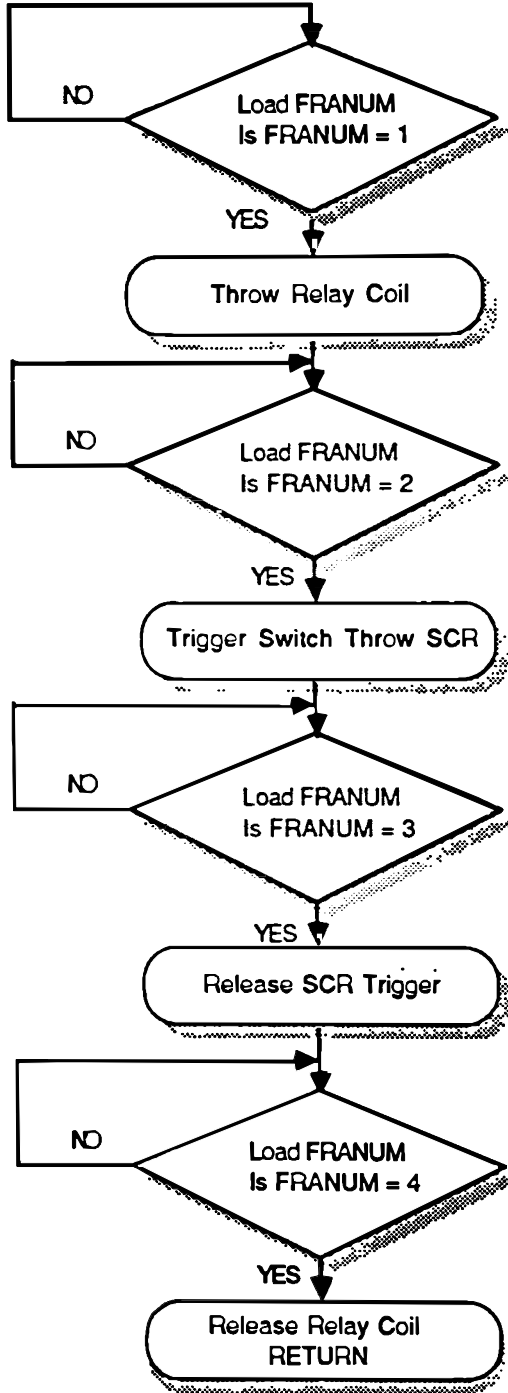
APPENDIX B

FLOW CHARTS

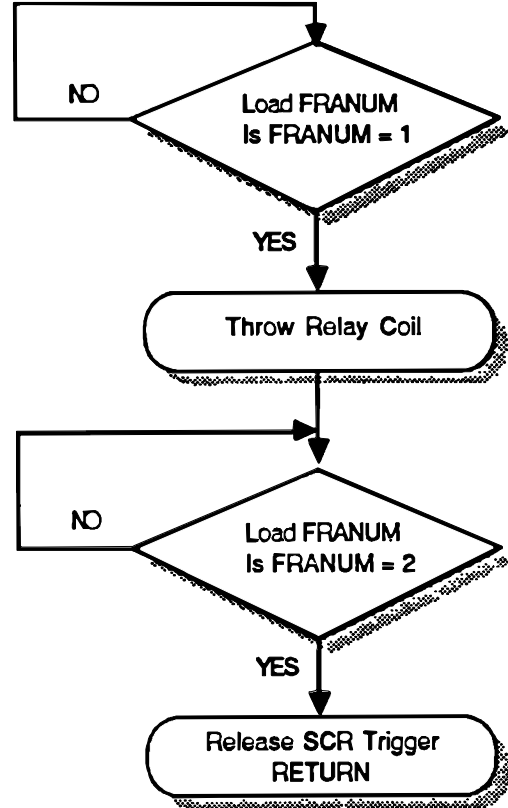
Main Program Flow



INLP, THGH Switching Procedures

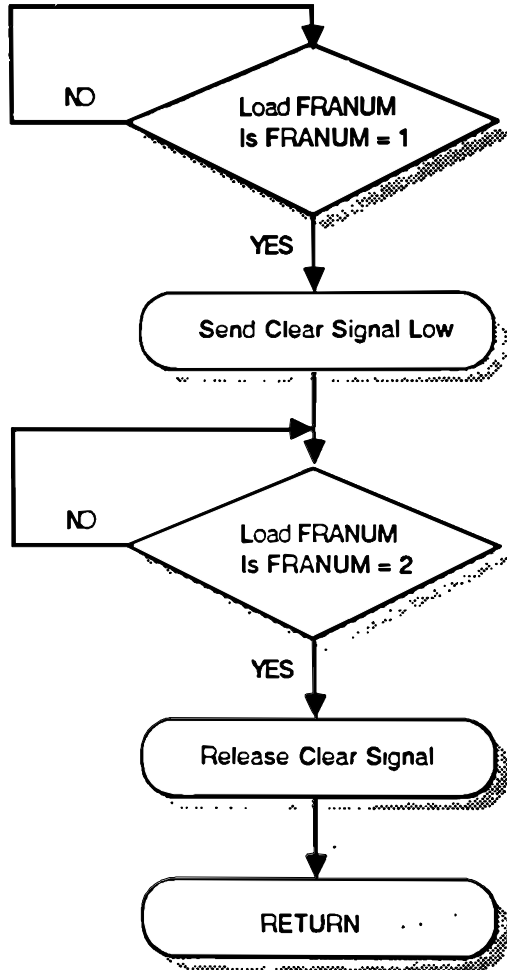


OUTL, ELEV Switching Procedures



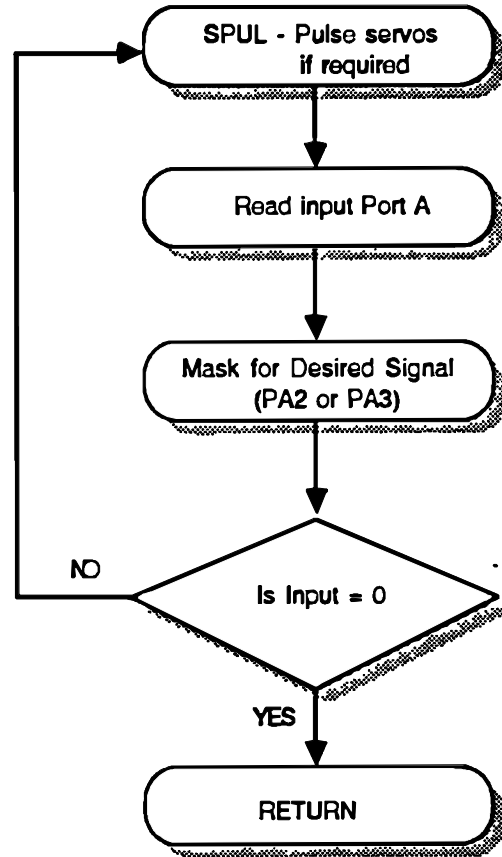
CLUP,CLDN,BKCL

Signal Clearing Procedures



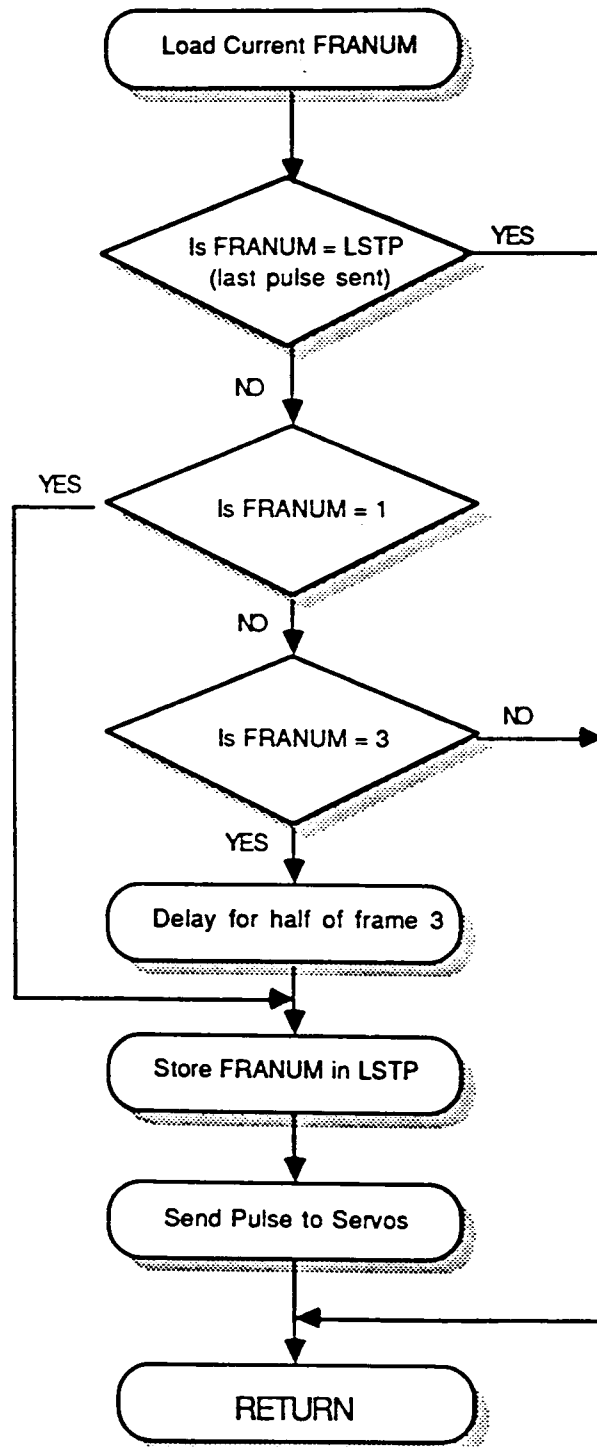
NBKR,SBKR

Block Clear Waiting Procedures



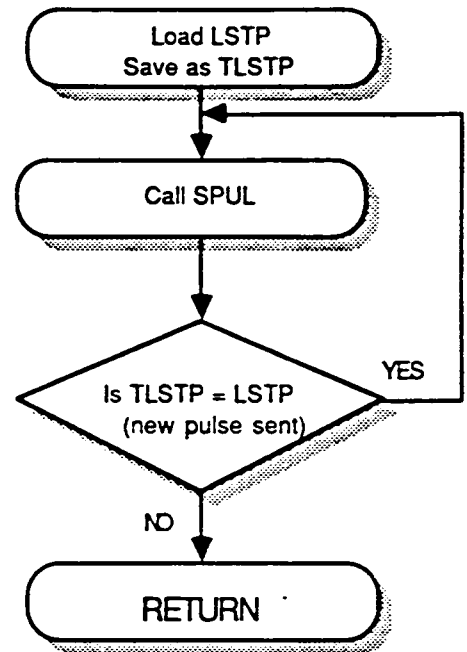
SPUL Subroutine

Pulse Servos if Start of Frame 1 or Middle of Frame 3

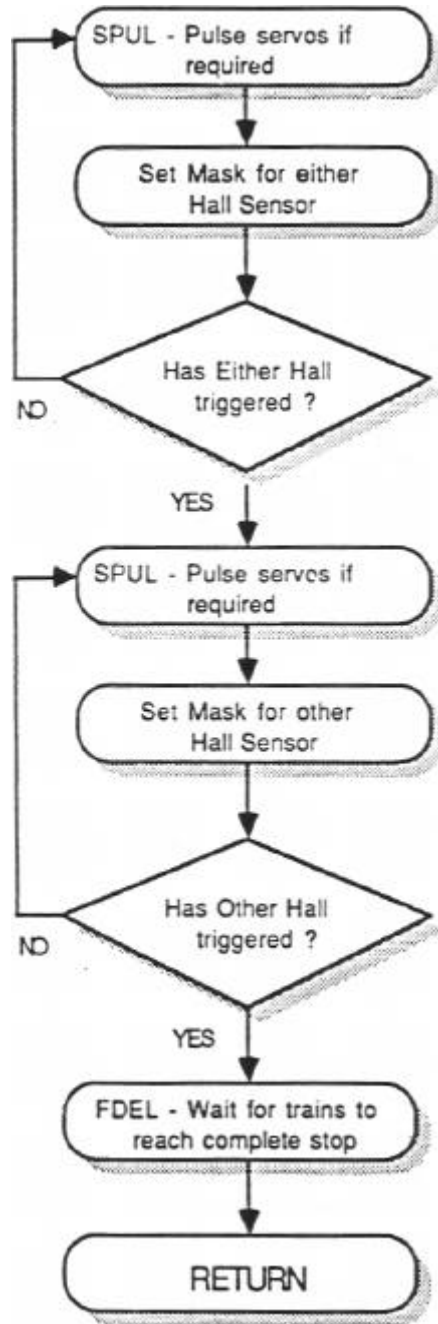


PULS Subroutine

Call SPUL until a Pulse is sent to Servos

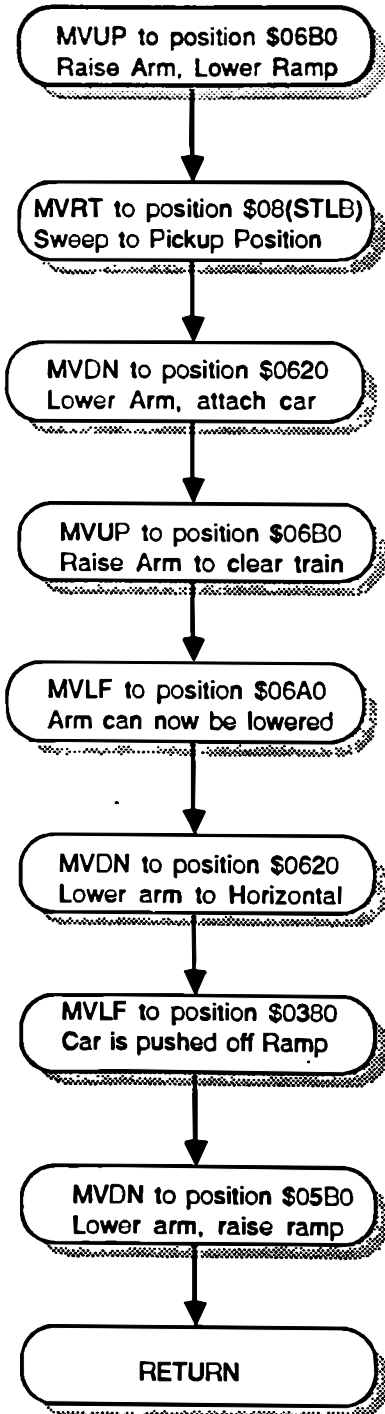


HWAIT Subroutine



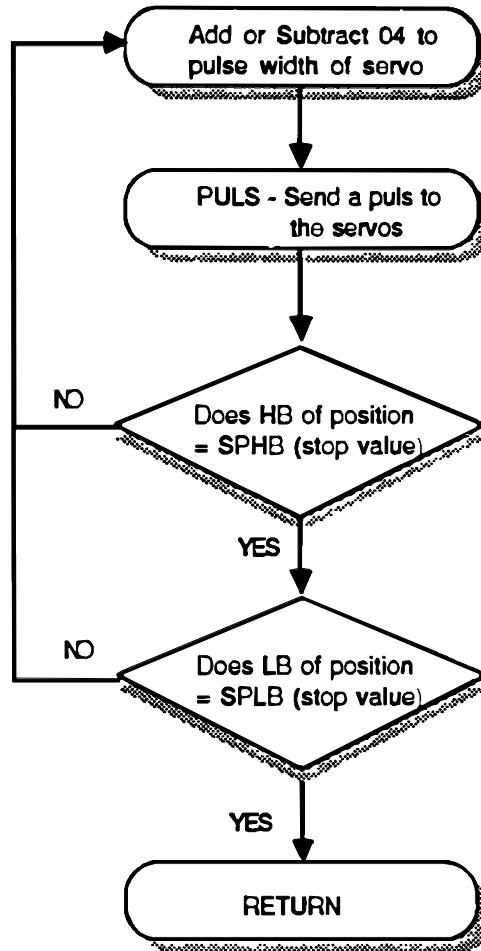
ARMG

Arm Motion Sequencing



MVUP, MVDN..

Arm Movement Procedures



APPENDIX C
CODE


```

; MJFINAL.ASM
; by Michael Majors and Jamie Stori
; May 24, 1992
; Sequencing Software for Tansportin Cars
;
;
%include LECKY3_2.ASM ; Must Run with LECKY v 3.2. Makes use of Timer
; 1 of $A000 VIA
;
; Versatile Interface Adapter VIA Addresses
;

V8ORB equ $8000 ; $8000 Data Register B - Used for Output
V8DRB equ $8002 ; $8000 Data Direction Register B
V8IRA equ $8001 ; $8000 Data Register A - Used for Input
V8DRA equ $8003 ; $8000 Data Direction Register B
V8ACR equ $800B ; $8000 Auxiliary Control Register
T8LB equ $8004 ; $8000 Timer 1 Low Byte
T8HB equ $8005 ; $8000 Timer 1 High Byte
TALB equ $A004 ; $A000 Timer 1 Low Byte
TAHB equ $A005 ; $A000 Timer 1 High Byte

; Local Variable Addresses
;

DISP equ $4000 ; TIL display address

VTLB equ $00 ; Low Byte of Vertical Servo Pulse Width
VTHB equ $01 ; High Byte of Vertical Servo Pulse Width
HTLB equ $02 ; Low Byte of Horizontal Servo Pulse Width
HTHB equ $03 ; High Byte of Horizontal Servo Pulse Width
STLB equ $04 ; Low Byte of Horizontal Stop Position

PSCT equ $05 ; Pulse Count - Count down for sending pulses
SPLB equ $06 ; Low Byte of Pulse Width to Stop Movement
SPHB equ $07 ; High Byte of Pulse Width to Stop Movement
HLMSK equ $09 ; Mask for reading Hall sensors
FDCT equ $10 ; Number of Frame cycles to delay for
LSTP equ $0A ; Frame Number of Last Servo Pulse
TLSTP equ $0B ; Temporary Storage of LSTP

;
; Initialization
;

org $F000
sei ; set interrupt disable
cld ; clear decimal mode
ldx #$FF
txs ; set stack pointer all 1's
lda #09
sta BLKID ; set computer #9
jsr INIT ; initialize Lecky routine
lda #$FF

```

```
    sta V8DDRb ; set $8000 Port B to output mode
    lda #$00
    sta V8DDRA ; set $8000 Port A to input mode
    lda #$FF
    sta V8ORB  ; Set all outputs on Port B high
    lda #$80
    sta V8ACR  ; set VIA $8000 timer #1 to one-shot mode
    lda VACR   ; store VIA $A000 Auxiliary Control Register
    and #$3F  ; clear bits 6 and 7 of VACR
    clc
    adc #$80   ; set bit 7 of VACR
    sta VACR   ; Set VIA $A000 timer #1 to one-shot mode

    jsr STRT   ; Move Arm to Ramp Elevated Position

;
; Begin Sequencing
;
BEGIN lda #$22 ; Screen Message 22 - Initialization Complete
      sta DISP

      jsr ELEV  ; Throw Switches to Elevated Track
      jsr INLP  ; Throw Switches to go In Loop
      jsr CLDN  ; Clear Lower Track Kill
      jsr CLUP  ; Clear Upper Track Kill

      jsr HWAIT ; Wait for both trains to stop past sensors

      jsr CLUP  ; release Upper Train
      jsr CLDN  ; release Lower Train

      jsr HWAIT ; Wait for boths trains to stop past sensors
              ; cars now aligned for loading

      lda #$C0 ; Store low byte of pickup position in STLB
      sta STLB
      jsr ARMG  ; Initiate arm motion

      jsr CLUP  ; Release Upper Train
      jsr CLDN  ; Release Lower Train

      jsr HWAIT ; Waits for boths trains to stop past sensors
              ; cars now aligned for loading

      lda #$90 ; Store low byte of pickup position in address STLB
      sta STLB
      jsr ARMG  ; Initiate second arm motion

      jsr BKCL  ; clear South train
      jsr CLUP  ; release upper train

      lda #$F7 ; Hold upper track kill clear to avoid retriggering
      sta V8ORB

      jsr NBKR  ; Wait for North train to clear Track Clear Sensor
```

```
    lda #$FF      ; Release upper track kill clear
    sta V8ORB

    jsr THGH      ; Set through switch position

    lda #$FF      ; Set North block override
    sta NOVRD

    jsr SBKR      ; Wait for South Train to clear

    jsr OUTL      ; Set OutLoop switch position

    jsr CLDN      ; Release lower train

    lda #$EF      ; Hold Lower Track kill clear to avoid retriggering
    sta V8ORB

    jsr NBKR      ; Wait for North Train to clear Track Clear Sensor

    lda #$FF      ; Release Lower Track Kill
    sta V8ORB

    lda #00       ; Remove North Block Override
    sta NOVRD

    jmp BEGIN     ; Rerun Routine

; End Of Sequencing Loop
; Subroutines Follow
;

                                ; HWAIT waits for both trains to trigger the Hall
                                ; Sensors in either order
HWAIT lda #$AA      ; Display AA - Waiting for First Triggering
      sta DISP

hold1 jsr SPUL      ; Send Pulse to Servos if Required
      lda V8IRA     ; Wait for Hall to trigger on either track
      and #$03      ; ( either PA0 or PA1 to go low )
      cmp #$03
      beq hold1

      cmp #$02      ; Set mask for other Hall sensor
      beq mas2
      lda #$01      ; Mask for PA0 if PA1 already triggered
      jmp mdone
      lda #$02      ; Mask for PA1 if PA0 already triggered

mdone sta HLMSK
      lda #$BB      ; Display BB - Waiting for Second Triggering
      sta DISP

hold2 jsr spul
      lda V8IRA     ; wait for Hall to trigger on other track
```

```

and HLMSK          masked bit to go low
cmp #$00
bne hold2

lda #$0A          ; wait for trains to stop (10 full frame cycles)
sta FDCT
jsr FDEL

rts

jsr FRA1          ; Set Switches to In Loop position
lda #$FE          ; Throw Relay Coil during FRANUM=1 (PB0)
sta V8ORB
jsr NEXT
lda #$FC          ; Pulse SCR during FRANUM=2 (PB1)
sta V8ORB
jsr NEXT
lda #$FE          ; Remove SCR pulse at FRANUM=3
sta V8ORB
jsr NEXT
lda #$FF          ; Release Relay Coil at FRANUM=4
sta V8ORB
rts

THGH jsr FRA1     ; Set Switches to Through position
lda #$FE          ; Throw Relay Coil during FRANUM=1 (PB0)
sta V8ORB
jsr NEXT
lda #$FA          ; Pulse SCR during FRANUM=2 (PB2)
sta V8ORB
jsr NEXT
lda #$FE          ; Remove SCR pulse at FRANUM=3
sta V8ORB
jsr NEXT
lda #$FF          ; Release Relay Coil at FRANUM=4
sta V8ORB
rts

OUTL jsr FRA1     ; Set Switches to Out Loop position
lda #$FD          ; Pulse SCR during FRANUM=1 (PB1)
sta V8ORB
jsr NEXT
lda #$FF          ; Remove SCR pulse at FRANUM=2
sta V8ORB
rts

ELEV jsr FRA1     ; Set Switches to Elev position
lda #$FB          ; Pulse SCR during FRANUM=1 (PB2)
sta V8ORB
jsr NEXT
lda #$FF          ; Remove SCR pulse at FRANUM=2
sta V8ORB
rts

CLUP jsr FRA1     ; Clear Upper Track Kill

```

```
    lda #$F7      ; Hold PB3 Low during FRANUM=1
    sta V8ORB
    jsr NEXT
    lda #$FF      ; Release Upper Track Kill
    sta V8ORB
    rts

CLDN jsr FRA1     ; Clear Lower Track Kill
    lda #$EF      ; Hold PB4 Low during FRANUM=1
    sta V8ORB
    jsr NEXT
    lda #$FF      ; Release Lower Track Kill
    sta V8ORB
    rts

BKCL jsr FRA1     ; Externally trigger South Track Clear
    lda #$DF      ; Hold PB5 Low during FRANUM=1
    sta V8ORB
    jsr NEXT
    lda #$FF      ; Release South Track Clear
    sta V8ORB
    rts

SBKR lda #$2C     ; Wait for South Block to Clear
    sta DISP      ; Display 2C - Waiting for S Train to Clear Sensor
sbk2 jsr SPUL     ; Pulse Servos if Required
    lda V8IRA
    and #$04      ; Mask input to PA2
    cmp #$00      ; Wait for PA2 to go low
    bne sbk2
    rts

NBKR lda #$1C     ; Wait for North Block to Clear
    sta DISP      ; Display 1C - Waiting for N Train to Clear Sensor
nbk2 jsr SPUL     ; Pulse Servos if Required
    lda V8IRA
    and #$08      ; Mask input to PA3
    cmp #$00      ; Wait for PA3 to go low
    bne nbk2
    rts

FRA1 lda #01      ; Wait for FRANUM=1 to Occur
    cmp FRANUM
    bne FRA1
    jsr SPUL      ; Pulse Servos
    rts

NEXT lda FRANUM   ; Wait for FRANUM to change value
nex2 cmp FRANUM
    beq nex2
    jsr spul
    rts

FDEL jsr FRA1     ; Delay for number of frame cycles stored in FDCT
    jsr NEXT
    ldx FDCT
```

```

dex
stx FDCT
stx DISP      ; Display countdown of FDCT
cpx #00
bne FDEL
rts

ARMG lda #$60      ; Motion of Arm - Transfer Car from High to Low
Track
      sta DISP      ; Display 60 - Initiation of Arm Motion

      lda #$06      ; Load Stopping Value #$06B0
      sta SPHB
      lda #$B0
      sta SPLB
      jsr MVUP      ; Move Up to Stop Value

      lda #$08      ; Load Stopping Value #$08 (STLB)
      sta SPHB      ; STLB varies for each of the two Pickups
      lda STLB
      sta SPLB
      jsr MVRT      ; Move Right to Stop Value

      lda #$06      ; Load Stopping Value #$0620
      sta SPHB
      lda #$20
      sta SPLB
      jsr MVDN      ; Move Down to Stop Value

      lda #$06      ; Load Stopping Value #$06B0
      sta SPHB
      lda #$B0
      sta SPLB
      jsr MVUP      ; Move Up to Stop Value

      lda #$06      ; Load Stopping Value #$06A0
      sta SPHB
      lda #$A0
      sta SPLB
      jsr MVLF      ; Move Left to Stop Value

      lda #$06      ; Load Stopping Value #$0620
      sta SPHB
      lda #$20
      sta SPLB
      jsr MVDN      ; Move Down to Stop Value

      lda #$03      ; Load Stopping Value #$0380
      sta SPHB
      lda #$80
      sta SPLB
      jsr MVLF      ; Move Left to Stop Value

      lda #$05      ; Load Stopping Value #$05B0
      sta SPHB
      lda #$B0
```

```
    sta SPLB
    jsr MVDN      ; Move Down to Stop Value

    rts

    lda LSTP      ; PULS routine calls SPUL until a Pulse has been
                  ; to the servos
    sta TLSTP     ; Store LSTP temporarily
    jsr SPUL
    lda LSTP      ; Compare LSTP with TLSTP until different
    cmp TLSTP
    beq PULS
    rts

    lda FRANUM    ; SPUL sends pulses to the two servos at the start
                  ; of Frame 1 and in the middle of Frame 3
                  ; The frame that a signal was last sent out on is
                  ; Stored in LSTP to allow for such a pattern

    cmp LSTP
    beq sdon
    cmp #01
    beq sdo       ; IF LSTP=3 (and FRANUM=1), jump to send pulse
    cmp #03
    beq sdo3
    jmp sdon

sdo3  ldy #00     ; If LSTP=1 (and FRANUM=3), wait until middle of F3
loo8  ldx #00
loo9  inx
      cpx #$DE
      bne loo9
      iny
      cpy #$06
      bne loo8

sdo   sta LSTP    ; Store new value of LSTP, and pulse servos with
                  ; values stored in VTLB,VTHB,HTLB,HTHB
      lda VTLB
      sta T8LB
      lda VTHB
      sta T8HB
      lda HTLB
      sta TALB
      lda HTHB
      sta TAHB
      ldy #00
sdon  rts

MVDN jsr DOWN     ; Move Down until Vertical Servo is at stop position
      jsr PULS
      lda VTHB
      cmp SPHB
      bne MVDN
      lda VTLB
      cmp SPLB
      bne MVDN
```

```
    rts

MVUP jsr UP          ; Move Up until Vertical Servo is at stop position
     jsr PULS
     lda VTHB
     cmp SPHB
     bne MVUP
     lda VTLB
     cmp SPLB
     bne MVUP
     rts

MVRT jsr RGHT       ; Move Right until Hor Servo is at stop position
     jsr PULS
     lda HTHB
     cmp SPHB
     bne MVRT
     lda HTLB
     cmp SPLB
     bne MVRT
     rts

MVLF jsr LEFT       ; Move Left until Hor Servo is at stop position
     jsr PULS
     lda HTHB
     cmp SPHB
     bne MVLF
     lda HTLB
     cmp SPLB
     bne MVLF
     rts

UP   clc            ; Two Byte Addition of #$04 to pulse width of
     ; Vertical Motion Servo
     lda VTLB
     adc #$04
     sta VTLB
     lda VTHB
     adc #00
     sta VTHB
     rts

DOWN sec           ; Two Byte Subtraction taking #$04 from pulse width
     ; of Vertical Motion servo
     lda VTLB
     sbc #$04
     sta VTLB
     lda VTHB
     sbc #00
     sta VTHB
     rts

RGHT clc          ; Two byte addition of #$04 to pulse width of
     ; Horizontal Motion Servo
     lda HTLB
     adc #$04
```



```
    sta HTLB
    lda HTHB
    adc #00
    sta HTHB
    rts

LEFT sec          ; Two byte subtraction taking #$04 from pulse width
                  ; of Horizontal Motion Servo
    lda HTLB
    sbc #$04
    sta HTLB
    lda HTHB
    sbc #00
    sta HTHB
    rts

                ; Subroutine to move Servos to starting position
STRT lda #$07    ; Load High-Byte Value For Vertical Servo
    sta VTHB
    lda #$00    ; Load Low-Byte Value for Vertical Servo
    sta VTLB
    lda #$03    ; Load High-Byte Value for Horizontal Servo
    sta HTHB
    lda #$80    ; Load Low-Byte Value for Horizontal Servo
    sta HTLB

    lda #$11    ; Screen message 11 - Program Initialization
    sta DISP

    lda #00    ; Clear Pulse Count
    sta PSCT

armi jsr PULS    ; Send #$10 pulses to servos, allow them to achieve
                ; their starting position
    ldx PSCT
    inx
    cpx #$10
    stx PSCT
    bne armi

    lda #$05    ; Move arm down, lift ramp
    sta SPHB
    lda #$B0
    sta SPLB
    jsr MVDN

    rts

; Set Reset Vector to Start of Program
;
    org $fffc
    dw $f000

end
```