LAB 6: THE CALCULATOR

THE PURPOSE OF THIS PROJECT IS TO BUILD A 4 DIGIT, INTEGER, STACK-BASED CALCULATOR THAT ADDS & SUBTRACTS.

GENERAL DESIGN.

THE SYSTEM WILL HAVE MAJOR COMPONENTS:

- KEYBOARD DECODE: THIS DEBOUNCES THE KEY MATRIX AND DECODES IT INTO THE NECESSARY DATA & CONTROL SIGNALS.

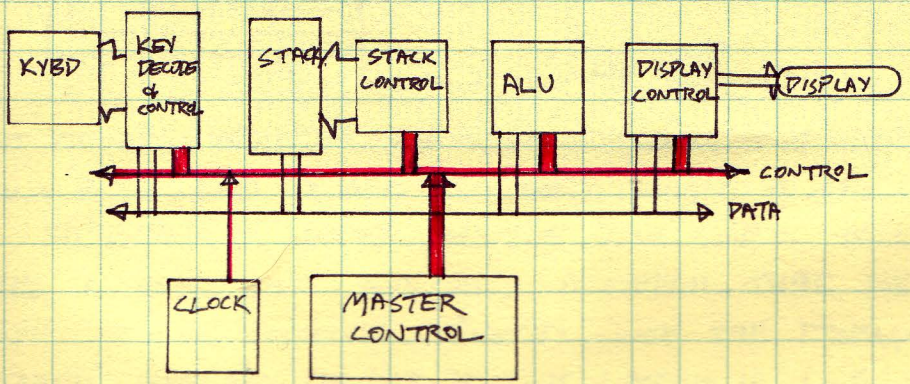
- A.L.U.: PROVIDES NECESSARY FUNKTIONS OF ADDING, SUBTRACTING AND COMPARING NUMBERS. IMPLEMENTATION WILL (PROBABLY) CONSIST OF PROM) LOGIC ~~EAF~~ WORKING SERIALLY, ONE DIGIT AT A TIME.

- DISPLAY DRIVER: READS DATA FROM THE X-REGISTER, DECODES & MULTIPLEXS IT FOR DISPLAY ON THE LED's

- STACK: ~~LO~~ CONTAINS THE 4 INTERMEADIT VALUES AND ASSOCIATED LOGIC FOR PUSHING AND POPPING ELEMENTS, ALONG WITH LATCHES FOR SIGN BITS.*

- MASTER CONTROL: HOUSES MICRO CODE FOR SEQUENCING AND CONTROL OF ALL OPERATIONS.

- BUSS & CLOCK: SYNCHRONIZES ALL ELEMENTS AND INTERCONNECTS THEM

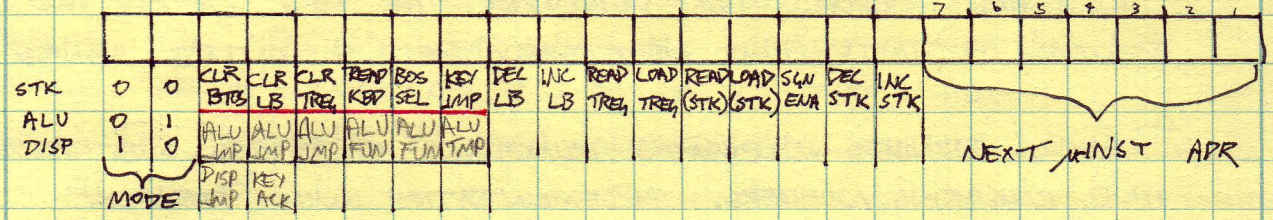


*LATCHES WERE ELIMINATED LATER IN DESIGN

PRELIMINAY μWORD

SYSTEM CONTROL WILL BE DRIVEN MOSTLY BY MICROCODE IN THE MASTER CONTROL UNIT. THE BUSS CONSISTS OF A 4BIT; BCD DATA PATH AND NECESSARY CONTROL LINES, DRIVEN BY MICRO CODE IN EPROM.*

THE FOLLOWING IS A LAYOUT OF THE MICRO CONTROL WORD:



AS THE μ PROGRAM RUNS, THE ADDR OF THE NEXT INST IS LATCHED, AND ON THE NEXT CYCLE IS USED AS THE PROM ADR.

AS THE PROGRAM RUNS, THE FOLLOWING SIGNALS CONTROL THE MACHINE:

INC STK	INCREMENTS CURRENTLY SELECTED STACK ADR REG (COUNTER)
DEC STK	DECREMENTS " " " " " "
SGN ENA	ADDRESSES THE SIGN (VS. DIGIT) STACK REGISTER, ENABLES COMPLIMENTOR
LOAD (STK)	LOADS THE STACK AT ADR OF CURRENT STACK REG FROM THE BUS
READ (STK)	READS " " " " " " " " ONTO " "
INC LB	INCREMENTS THE "LOWER BITS" (DIGIT) ~~ADDRESS REGISTER~~ & COUNTER
DEC LB	DECREMENTS " " " " " "
KEY JMP	'OR'S FUNCTION KEY CODE ONTO NEXT INST BUS (JUMP)
BOS SEL	SELECTS BOS AS CURRENT STK ADR IF 1, TOS IF Ø
READ KBD	READS KEYBOARD DATA ONTO BUS
CLR TREG	SETS TMP REGISTER (TREG) TO ALL ZERO'S
CLR LB	SETS "LOWER BITS" REG TO Ø
CLR BTOS	RESETS BOS OR TOS REG TO INITIAL VALUE

* DEVELOPMENT CONTINUES THIS WEEKEND.... ~~NOT~~ WATCH THIS SPACE FOR FURTHER DETAILS

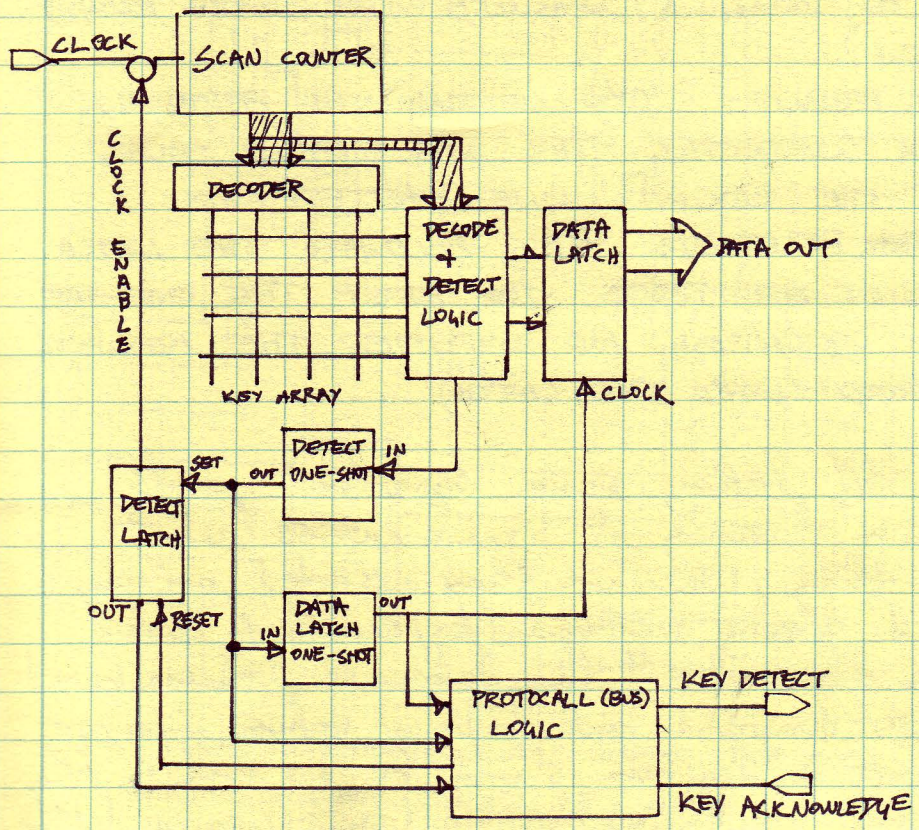
CALCULATOR - MODULE DESIGN NOTES. Fri 2/2 JK

CLOCK:

THE CLOCK WAS PRIMARILY STOLEN FROM LAB 5, CONSISTING OF A PUSH BUTTON SINGLE STEP AND A FREE RUNNING CLOCK (EXACT FREQ TO BE DETERMINED. A MASTER-CLEAR AND UTILITY SWITCH ARE ALSO PROVIDED (SEE NOTES BELOW ON CLOCK SPEED SELECTION) CLOCK SPEED IS SET TO A UNIFORM SQUARE WAVE WITH A 72mS PERIOD

KEYBOARD (REFER TO SCHEMATIC)

THE AVAILABLE KEYBOARD CIRCUIT ONLY PROVIDES ROW & COLUMN INPUTS, WITH THE KEYS PROVIDING SHORTS AT THE INTERSECTIONS. THE BLOCK DIAGRAM IS:



THE CIRCUIT WORKS BY SEQUENTIALLY PULSING THE ~~COLUMN~~ COLUMN LINES USING A DECODED 2 BIT COUNTER. WHEN A KEY IS PRESSED, THIS IS DETECTED BY GATES ON THE ROW LINES. THIS LOGIC THEN FIRES A ONE SHOT AND SETS A LATCH THAT DISABLES THE CLOCK INPUT TO THE COUNTER. (NOTE: SINCE THE PROPAGATION DELAY BETWEEN THE KEY DETECT & THE LATCH IS 1μS, THE CLOCK HAD TO BE SLOWED TO HAVE A LONGER WAVELENGTH THAN THAT, TO PREVENT INCREMENTING PAST THE DETECTED KEY). THE FIRST ONE SHOT PROVIDES A DELAY OF 20 ms TO ALLOW THE SWITCH BOUNCE TO SETTLE.

(CALCULATOR)

KEYBOARD LOGIC, CONT.

WHEN THIS ONE SHOT IS FINISHES, ANOTHER IS FIRED TO CLOCK THE DATA INTO THE OUTPUT LATCH. THE CLOCK DISABLE LATCH ALSO SERVES AS A FLAG TO INDICATE DATA IS AVAILABLE TO THE BUS. THIS LATCH REMAINS SET UNTIL AN ACKNOWLEDGEMENT IS RECIEVED FROM THE CONTROL BUSS. ~~THE~~ WHEN KEY_ACK IS RECIEVED (AND IF THE BUTTON IS RELEASED AND THE ONE SHOTS ARE FINISHED) THE LATCH WILL RESET, ENABLING BOTH THE COUNTER AND THE ONE-SHOTS.*

IMPLEMENTATION:*

BOTH ONE-SHOTS USED 10μF CAPS AND 2KΩ RESISTORS TO PROVIDE A DELAY OF 20ms. (I MEASURED THE SWITCH BOUNCE AT AROUND 8-15ms).

MAJOR CONSRUCTION PROBLEM WAS GETTING THE PROTOCALL CIRCUIT TO WORK CORRECTLY. THIS WAS FINALLY SOLVED BY TURNING ON THE KEY_HIT LINE -AFTER- THE BUTTON WAS ~~BEING~~ RELEASED. THIS ALLOWED THE LATCH /CLOCK-ENABLE CIRCUIT ~~FREE~~ TO KEEP THE ONE-SHOT OFF, PREVENTING OSCILATION, AND SIMPLIFIED THE KEY_ACK CIRCUIT (IT MERERLY CLEARS THE LATCH)

OK - but still lacking some important parts
- need a glossary of signal names (what are they, when are they asserted, and why)
- need timing diagrams (especially to show the elimination of the debounce problem here)
- logic diagrams need to be on vellum of course
- flow in diagram should be as much in the same direction as possible (your's is a little ~~too~~ random)
- label lines more on diagram (you do this pretty well already)
- need buffers between decoder and column lines (not obvious)

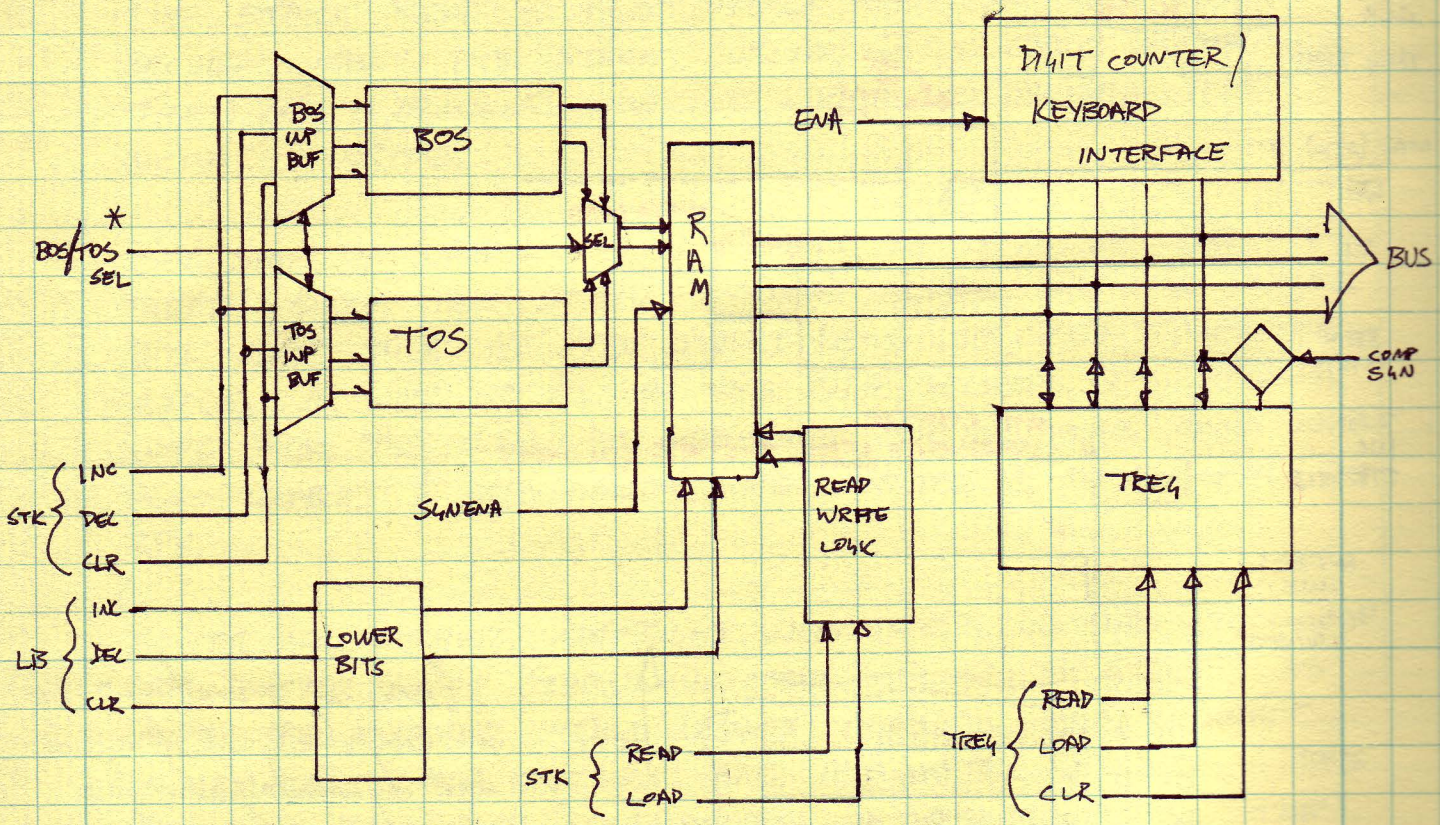
Web	1.75
Fri	2.0
Design	2.9
Doc	1.0
	7.65

*EDITORIAL NOTE: I HAD TO SCROUNGE MY OWN RESISTORS & CAPS TO BOTH SLOW THE CLOCK DOWN AND IMPLEMENT THE ONE SHOTS... THIS SCHEME WAS MUCH EASIER TO DEBUG THAN THE COUNTER-CHAIN

PLEASE
BOS - DON'T WRITE ON THE WALL

STACK CONTROL

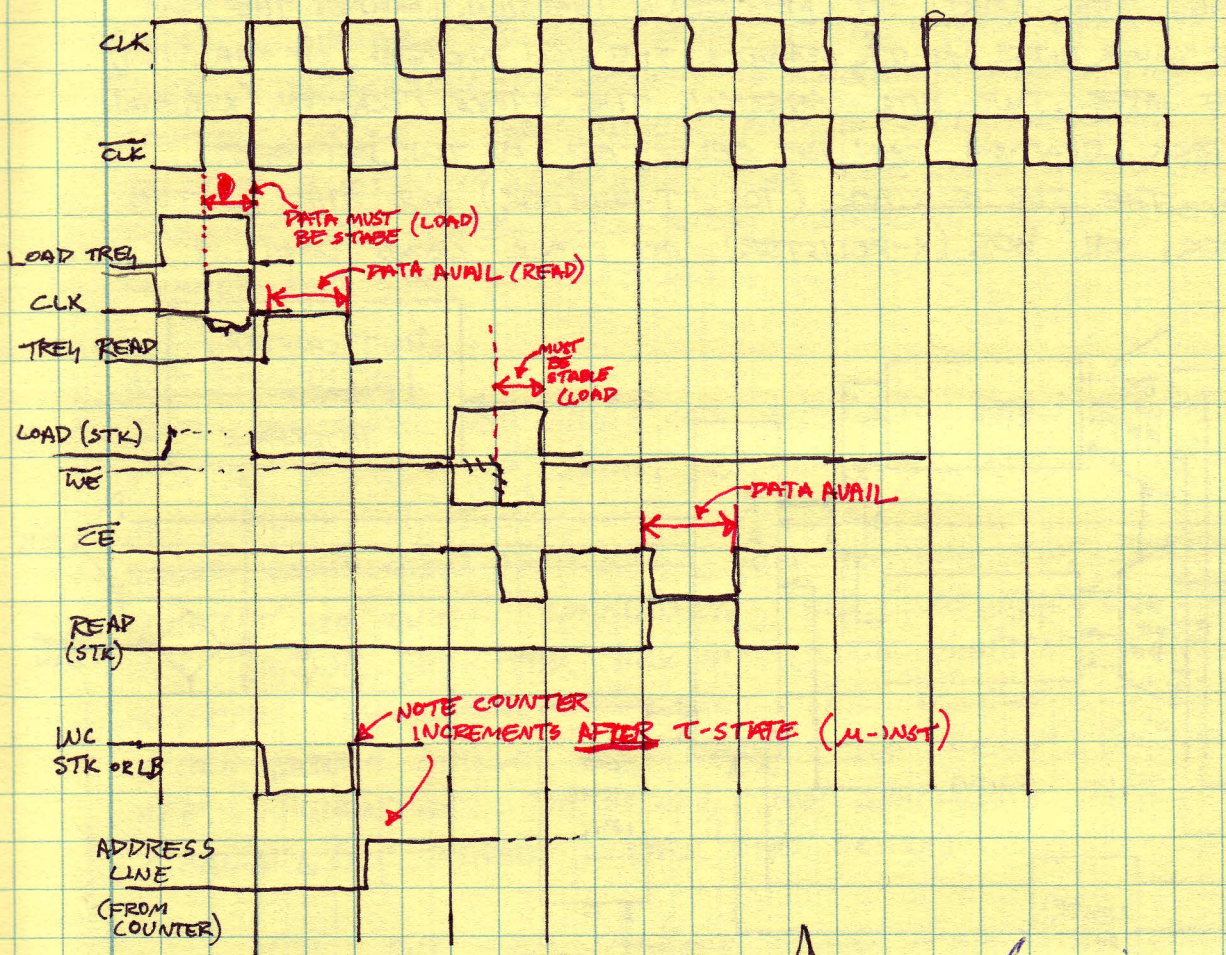
THE STACK SYSTEM USES A 2114 AS THE MAIN STORAGE ELEMENT. DATA IS STORED AS DIGITS, WITH THE BOTTOM TWO BITS SELECTING THE DIGIT, THE NEXT BIT SELECTING THE SIGN, (ASSUMES THE TWO DIGIT, OR "LOWER BITS" ARE 0) SELECTS THE SIGN INSTEAD OF THE DIGIT) AND THE UPPER TWO BITS SELECTING THE STACK LOCATION. (X, Y, Z or T) THE STACK LOCATION CAN BE SELECTED BY TWO DIFFERENT COUNTERS, THE TOP OF STACK (TOS, "T-REGISTER") AND THE BOTTOM OF STACK, OR BOS (X-REGISTER) (MY STACK GROWS DOWN...)

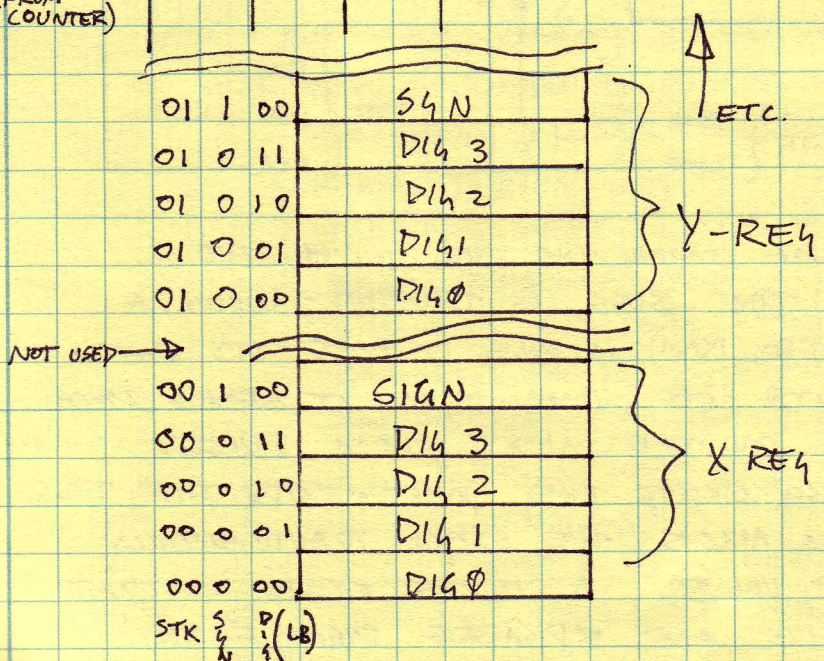


THE SIGN IS CHANGED BY LOADING THE TREG WITH THE SGNENA SIGNAL ON. THE BUSS IS TRISTATE; WHEN A DEVICE'S READ (KEYBD, T-REG, RAM) IS HIGH, IT PUTS ITS DATA ON THE BUSS. WHEN ITS LOAD SIGNAL IS ON, IT READS DATA FROM THE BUSS IN ON THE NEXT CLOCK PULSE. SOME MAGIC LOGIC ALSO EXISTS THAT INCREMENTS THE T.O.S. IF BOS = TOS. THIS ALLOWS THE STACK TO AUTOMATICLY "THROW AWAY" THE TOP VALUE. A COUNTER EXISTS TO COUNT THE NUMBER OF DIGITS AND DISABLE DIGIT ENTRY IF MORE THAN FOUR DIGITS ARE TYPED BEFORE LB. IS CLEARED. BY THE STATE MACHINE

*SIGNAL DEFS ON PAGE 28

TIMING FOR STACK MEMORIES





STACK MAP

Come in sometime so we can discuss your design and documentation. Your points will be assigned after the discussion.

RL

ckpB 2+2 3.0
Design 1.5
Doc 8.5/10
Δ CHANGE TO 2+2 3.0 1.0 8.0/10

STACK - CON'T

TIMING FOR THE TREG IS FAIRLY SIMPLE, A READ TREG ENABLES THE OUTPUTS OF THE 74125 LATCH ONTO THE BUS. FOR WRITE, THE LOAD TREG SIGNAL IS ANDED WITH THE INVERSE CLOCK, SO THE CLOCKING EDGE IS A ½ CYCLE DELAYED, ALLOWING 36μS FOR THE INPUTS TO SETTLE (NOTE: IF ~~LT~~ LOAD TREG & CLK BECOME HIGH AT ONCE (PROBABLY NOT POSSIBLE) A SPIKE MIGHT CLOCK JUNK INTO THE TREG, BUT WHO CARES SINCE THIS IS OVERWRITTEN A ½ CYCLE LATER)

THE STACK READ IS ALSO FAIRLY EASY, $\overline{CE}$ IS SIMPLEY BROUGHT LOW. FOR STACK WRITE, $\overline{WE}$ IS FIRST BROUGHT LOW, THEN $\overline{CE}$ IS BROUGHT LOW ½T LATER TO PERFORM THE ACTUAL WRITE.

THE SIGN IS STORED IN THE STACK AS BIT0 OF THE "100" DIGIT. SINCE THE STATE MACHINE ALWAYS HAS THE LOWER BITS SET TO 0 AT THE START & END OF EACH FUNCTIONAL CYCLE, A LINE TO A_2 OF THE 2114 SELECTS THE PROPER WORD. FOR THE SIGN. AS MENTIONED EARLIER, THIS SAME ENABLE ALSO ENABLES A COMPLIMENTOR WHEN THE TREG IS LOADED, FOR CHS OPS.

DIGITS ARE STORED ~~LEFT~~RIGHT JUSTIFIED IN THE STACK, AND ARE SHIFTED LEFT (THRU THE TREG W/ LOADS & STORES) AS MORE ARE ENTERED. THIS LEAVES THEM IN THE CORRECT ALLIGNMENT FOR ENTER'S AND ARITHMATK. PUSHES & POPS ARE DONE BY SIMPLEY INCREMENTING OR DECREMENTING THE BOS & TOS COUNTERS (TOS IS ONLY PARTIALLY USEFUL FOR COPYING DOWN THE STACK, BUT IT'S TOO LATE TO RIP IT OUT NOW. SOUNDED GOOD AT THE TIME ....). SELECTOR LOGIC DECIDES BOTH WHICH ONE GETS THE CONTROL LINES & WHICH ONE RUNS THE RAM ADDRESS LINES. THE DISPLAY WILL BE RUN BY THE BOS REGISTER, SINCE IT POINTS TO X DURING IDLE (NON-FUNCT) STATES

IMPLEMENTATION NOTES: MAJOR BUGS WERE WIRING THE BUS BACKWARDS & BEATING ON THE TWO (ACTUALLY FOUR) PULL DOWN RESISTORS TO WORK RIGHT. ~~AND~~ ERRORS IN THE RAM WRITE & BOS=TOS LOGIC WERE ALSO CORRECTED. (MOSTLY WRONG POLARITY). HAYES THINKS THE TREG WRITE TIMING IS WRONG (SHOULD USE NAND INTO CLK) BUT I DON'T BELIEVE HIM. WE'LL SEE....

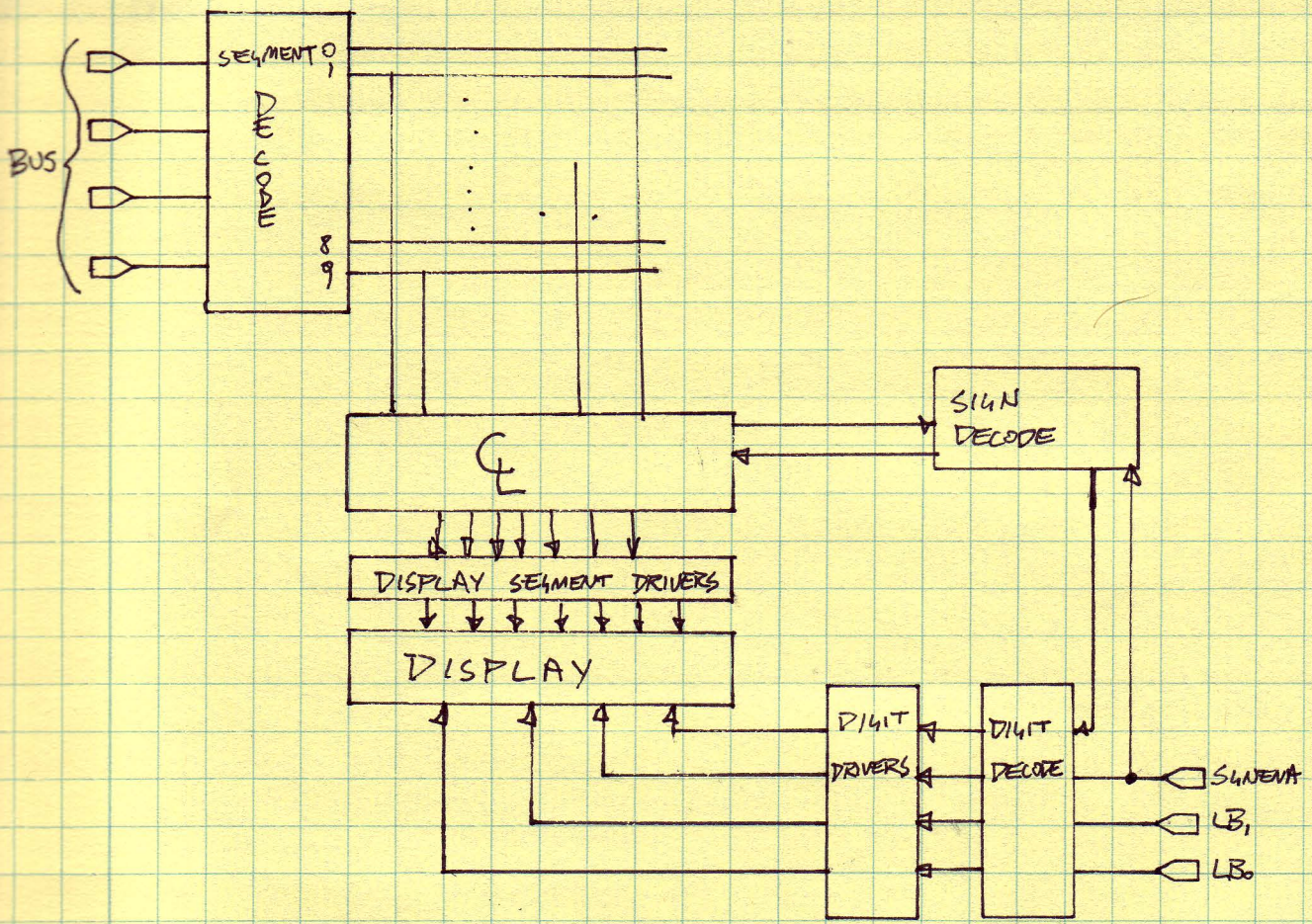
LED Display working Wed 2pm R.C.
ALL added subtract 2/2 SK

DISPLAY

THE DISPLAY IS A SEVEN SEGMENT LED TYPE WITH MULTIPLEXED SEGMENTS, AND 5 DIGIT ENABLE LINES. THE SEGMENTS ARE DRIVEN BY 74907 DRIVERS AND TURN THE SEGMENTS WITH A HIGH INPUT, AND THE DIGIT ENABLES ARE DRIVEN BY 74C906 DRIVERS AND TURN THE DIGIT ON WITH A LOW INPUT. THE DIGIT DRIVERS ARE SELECTED WITH A 74C42 DECODER. THE LOWER TWO BITS ARE DRIVEN DIRECTLY BY THE STACK LOWER BITS (DIGIT SELECT) INPUTS. (SEE P. 31). THE SEGMENTS ARE DRIVEN BY A C42 DECODER DRIVEN DIRECTLY FROM THE BUS (NO LATCH IS USED BECAUSE THE SYSTEM CLOCK IS SLOW ENOUGH TO ALLOW THE LED'S TO TURN ON). THE DECODER OUTPUT IS IN TURN FED TO COMBINATIONAL LOGIC TO DECODE THE DIGIT SYMBOLS. THE LOGIC CONSISTS OF OR GATES HOOKED UP TO THE RESPECTIVE DIGIT LINES THAT WILL TURN THAT SEGMENT OFF (EXCEPT FOR SEGMENT 'e', WHICH GETS TURNED ON). THIS INVERTED LOGIC SYSTEM ALLOWS SMALLER #'S OF INPUTS ON THE GATES. THE LAST TRICK IS THE SIGN. THE STATE MACHINE ONLY TURNS ON SGNENA WHEN LB=00, SO SGNENA IS JUST FED DIRECTLY TO THE C INPUT OF THE DECODER; SO LINE 4 GETS SELECTED WHEN SGNENA IS ON. IF THE SIGN IS NEGITIVE, A '-' IS DECODED AND FED TO THE DISPLAY. IF THE SIGN IS POSITIVE, THE D INPUT OF THE DIGIT SELECT DECODER IS TURNED ON, AND NONE OF THE DIGITS ARE SELECTED. LEAVING A BLANK.

dept 2
Design 1.5 1.2.5 Design is good (sign decode, etc.)
Doc Documentation is thorough, but
4.2.5 "blocky" — still needs a bit
more organization.
Bob

DISPLAY BLOCK DIAGRAM.



SINCE THE DISPLAY IS CONSTANTLY DRIVEN, NO EXPLICIT TIMING LOGIC IS NECESSARY. THE ONLY REQUIREMENT IS THAT LB BE ØØ DURING SGNENA HIGH. NORMALLY THE DISPLAY WILL BE DRIVEN WITH BOS SELECTED AND STACK READ TURNED ON. BY CYCLING THRU LB AND THEN TURNING ON SGNENA (WITH LB=00) THE FULL ~~DISPLAY~~ DISPLAY IS SHOWN. THIS WILL BE THE "IDLE LOOP" OF THE STATE MACHINE.

NOTE: ALU IS COMPLETE & WORKING BUT NOT DOCUMENTED YET. STAY TUNED

A.L.U.

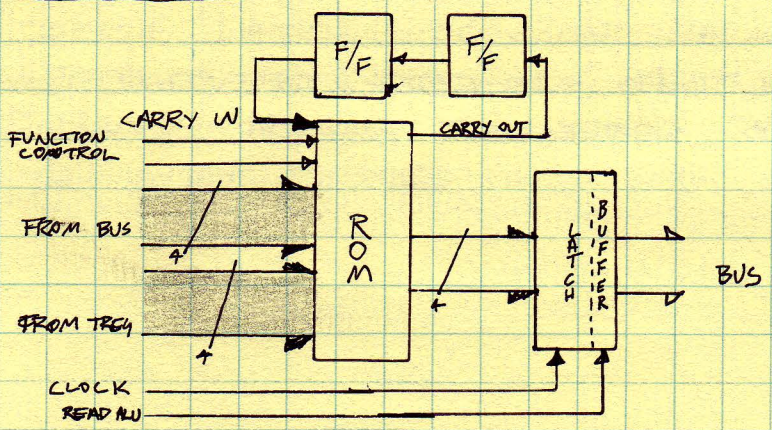
THE ARITHMATIC LOGIC UNIT PERFORMS ALL THE ARITHMATIC FOR THE CALCULATOR. IT IS IMPLEMENTED AS A LOOKUP TABEL USING ONE OF THE 2716 EPPROMS. THE LAYOUT FOR THE ADDRESS LINES IS GIVEN BELOW:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The prom used for the calculator is an Intel 2716 type.  It has 11 address
% lines and eight output lines, accessed as follows:
%
%   Output lines:
%      7  6  5  4  3  2  1  0
%     --+--+--+--+--+--+--+---
%     |xxxxxxx|CO|d3 d2 d1 d0|
%     ------------------------
%        V     V     V
%        |     |     +-------> Output Data (d3= MSB)
%        |     +-------------> Carry/Borrow Out (1= carry/borrow)
%        |                     also Compare Out (1= A greater)
%        +-------------------> Not used
%
% The input (address lines) are allocated as follows:
%
%
%      10 9  8  7  6  5  4  3  2  1  0
%     +--+--+--+--+--+--+--+--+--+--+---
%     |f1 f0|CI|a3 a2 a1 a0|b3 b2 b1 b0|
%     ----------------------------------
%        ^    ^     ^          ^
%        |    |     |          +------< B digit input (b3=MSB)(from Treg)
%        |    |     +-----------------< A digit input (a3=MSB)(main bus)
%        |    +-----------------------< Carry/Borrow In
%        +----------------------------< Function Code:
%                                        f1 f0    Function
%                                       -------  ----------
%                                        0  0    Add a to b
%                                        0  1    Subtract a from b (b-a)
%                                        1  0    Subtract b from a (a-b)
%                                        1  1    Compare a to b
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

THE A INPUT IS DRIVEN FROM THE BUSS, AND THE B INPUT IS DRIVEN FROM THE TREG. THUS, TO PERFORM AN OPERATION, THE FIRST NUMBER IS GATED INTO THE TREG AND THE SECOND IS READ ONTO THE BUS (USUALLY FROM THE STACK). THIS HAPPENS FOUR BITS AT A TIME, SO THE NEXT SET OF DIGITS MUST BE PROPAGATED THROUGH IN THE SAME ~~MANER~~ MANER.

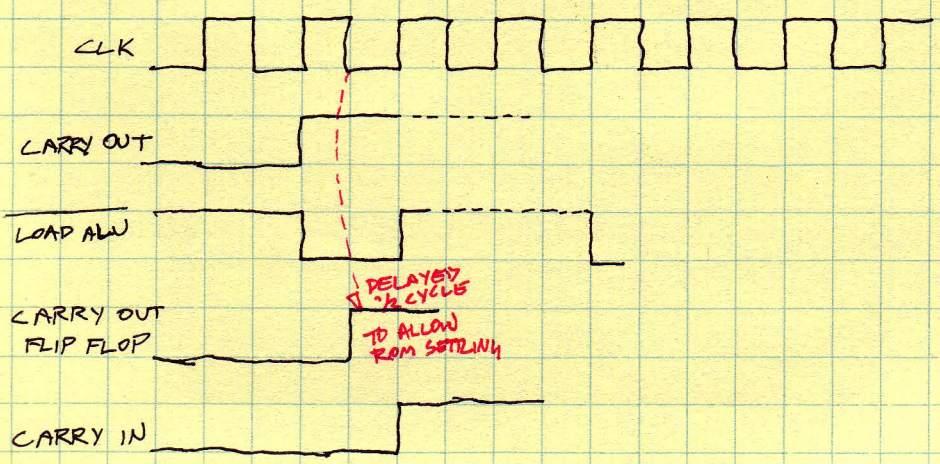
THE OUTPUTS OF THE ALU ARE LATCHED TO PREVENT "BOUNCE" WHILE THE DATA SETTELS, AND TO ALLOW THE ALU TO INPUT AND OUTPUT TO THE SAME BUS.

BLOCK DIAGRAM



THE CARRY IS PASSED INTO A FLIP/FLOP ON THE OUTPUT SIDE THAT LATCHES IT WHEN THE ALU IS LOADED. ON THE NEXT CLOCK CYCLE, THE CARRY IS FED BACK INTO THE ALU, TO FORM PART OF THE NEW ROM ADDRESS FOR THE NEXT DIGIT PAIR. THUS THE CARRY IS PROPAGATED WITHOUT ANY HELP FROM THE MASTER CONTROL.

TIMING



THE FLIP-FLOPS ARE RESET BY CLRLB TO ASSURE CARRY IS ZERO AT THE START OF A DIGIT SEQUENCE.

ALU - IMPLEMENTATION NOTES.

THE ALU WAS ENCODED WITH THE USE OF AN RLISP PROGRAM, MAKEALU.RED* THIS PROGRAM GENERATED THE NECESSARY OUTPUT TABLE AND PUT IT IN THE PROPER HEX FORMAT FOR THE PROM BLASTER.

THE FOLLOWING TABEL GIVES SOMEIDEA OF THE PLANNED ARITHMATIC SEQUENCES:

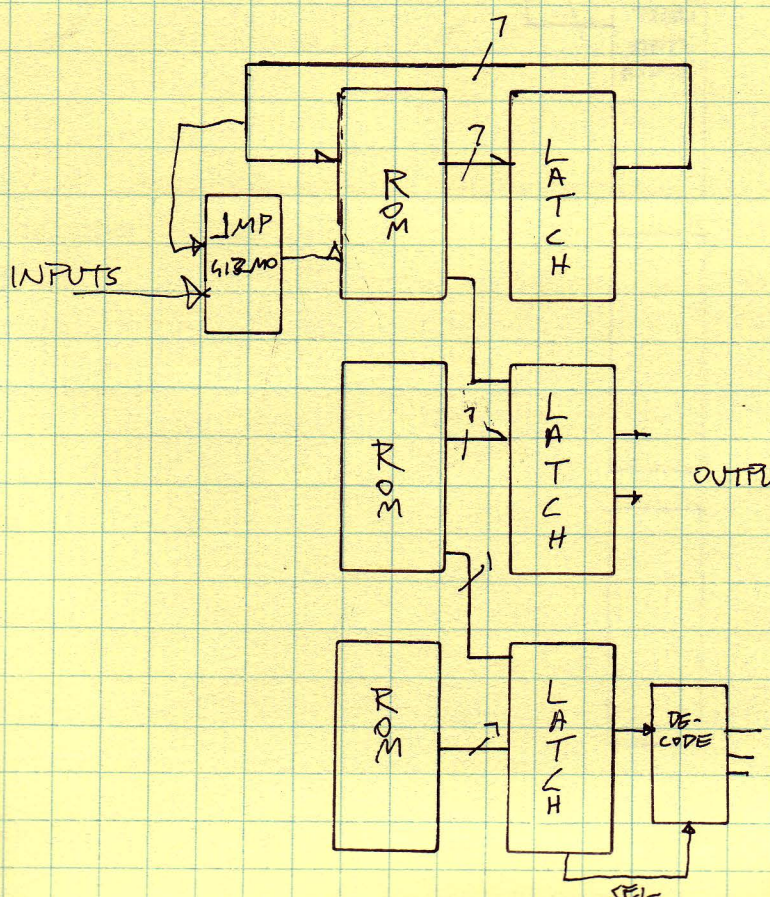
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The prom is used with the following scheme for arithmatic.  Since Trying
% to subtract big numbers from little ones produces incorrect results (no
% intermediate sign info), IT IS REQUIRED THE SMALLER BE TAKEN FROM
% THE LARGER
%
%                 Input Signs
% function       A         B              Action Taken
% --------- -------------         --- ----------------------------
%                +         +       (1)  ALU:= A + B;  Sgn:= +
%
%                +         -       (2)  if A >=B then ALU:= A - B, Sgn:= +
%                                       if A < B then ALU:= B - A, Sgn:= -
% Addition:
%                -         +       (3)  if A > B then ALU:= A - B, Sgn:= -
%                                       if A <=B then ALU:= B - A, Sgn:= +
%
%                -         -       (4)  ALU:=A + B, Sgn:= -
%   -------------------------------------------------------------
%                +         +            Same as (2), above.
%
%                +         -            Same as (1), above.
% Subtraction:
%                -         +            Same as (4), above.
%
%                -         -            Same as (3), above.
%
% To facilitate the compare function, some tricks are used.  First, the compare
% works from right to left, to maintain uniformity (and possibly microcode!)
% with the other functions.  The carry is used to move along the previous
% digit's result; so the same carry latch hardware can be used.  See GenValue
% (below) for the exact algorithm.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

*ABSOLUTLY NO RELATION TO ROMP. SEE LISTING IN NOTEBOOK

LAST MINUTE....

STATE MACHINE

THE STATE MACHINE IS IMPLEMENTED WITH THREE 2716 EPROMS, AND OUTPUT LATCHES. SEE THE NOTEBOOK PAGE "THE MICROWORD" FOR EXPLANATION OF THE SIGNAL DEFINITIONS. THE LOWER 7 BITS ARE USED AS THE "NEXT STATE" INPUT TO THE MACHINE. THEY ARE LATCHED BY A 74C374 LATCH. TO CLEAR THE MACHINE, THE TRI STATE OUTPUTS OF THE NEXT STATE LATCH ARE DISABLED, ALLOWING PULL DOWN RESISTORS TO BRING THE STATE ADDRESS TO Ø. FOR μ BRANCHES, SUCH AS CARRY OR KEYPRESS, THE APPROPRIATE JMP LINE IS BROUGHT HIGH, ALLOWING THESE LINES TO AFFECT THE NEXT STATE ADDR. (REFERED TO AS ROMUECTS IN THE μCODE ASM)



Very nice! and a distinct improvement in documentation!

It wasn't real clear, but I assume you run through one cycle as a compare, before the operation — using the ALU hardware.

I'd like to spend more time looking at your state machine set-up — the microcode in particular. I like the relationship of software to hardware. The innovative approach I think offsets the lack of more traditional documentation, (this time!)

Bob

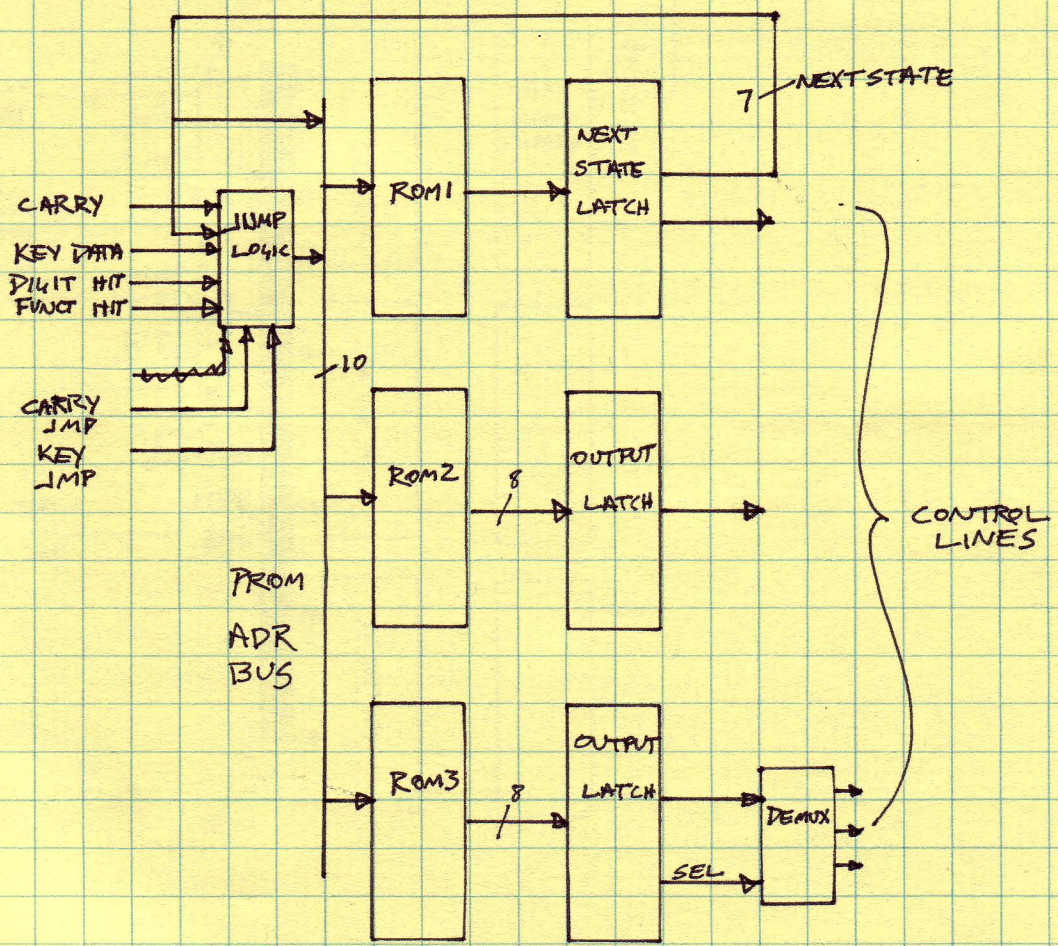
NOTE: I'LL RE-DO THIS NEXT WEEK, NO TIME NOW. SEE SCHEMATIC, MICROWORD, AND STATE MACHINE ASSEMBLY CODE & ASSEMBLER (RLISP) TO COMMENTED FOR MORE DOC.

dcpts 4
Design 3
Doc 3/10/10

THE STATE MACHINE

THE STATE MACHINE PROVIDES MOST* OF THE CONTROL FOR THE CALCULATOR. THE DESIGN IS BASICALLY A MICROCODED MACHINE WITH CONDITIONAL BRANCHING. THERE ARE SEVEN NEXT-STATE LINES, AND THREE ADDITIONAL INPUTS FOR DIGIT PRESSED, FUNCTION KEY PRESSED, AND CARRY OUT FROM ALU. WHEN THE DIGIT OR FUNCTION IS GATED IN TO THE ~~ALU~~ NEXT STATE. THIS ALLOWS THE MACHINE TO BRANCH TO A KNOWN LOCATION FOR A GIVEN KEY.

BLOCK DIAGRAM



*EXCEPT FOR BOS=TOS MASK & SIGN COMP (SEE P. 32) AND DISPLAY

Fri ckpt: Adds well. Stack works, except have to press CLX after ENT to enter a new number. Subtracts smaller numbers from larger, using "+" & CHS. Has a stack-roll key that works well. 1.4. 2.5/3 1.E. 11 Mar '83

Wed dept 2/2 BElem 9 March 83

NOTE: THERE SEEMS TO BE SOME CONFUSION ON HOW CHK POINTS WERE DELT WITH SINCE THE SCHEDUAL SLIPPED. BASICLY, I COMPLETED ALL CHK POINTS ON THE ORIGINAL SCHEDUAL, AND ALL BUT THE LAST ONE WORKED COMPLETLY.

STATE MACHINE, CONT

CONTROL LINE DECODING: THE MAJORITY OF THE CONTROL LINES* ARE FED DIRECTLY FROM THE OUTPUT LATCHES. THE EXCEPTIONS ARE: 1) THE COUNTER OUTPUTS, THESE ARE OR'ED WITH THE CLOCK. THIS ALLOWS THE COUNTER (LB OR STK) TO BE PULSED TWICE IN A ROW WITHOUT AN INTERMEDIATE STATE. #1) 2) TWO OF THE OUTPUT LINES ARE DECODED TO FOUR ON ROM 3. THE SELECT LINE (BIT 8) IS FED INTO A QUAD MUX HOOKED INTO BITS 4 + 5. WHEN THE LINE IS NOT SELECTED, A 'DEFAULT' VALUE IS FED INTO THE MUX. 3) A 'KEY ACK' IS GENERATED BY RAISING ALUF0, ALUF1, AND EXECUTING A READ KBD.

CONDITIONAL BRANCHING: A CONDITIONAL BRANCH IS DONE BY RASING THE DESIRED CONTROL LINE (KEY_JMP OR CARRY_JMP). THIS GATES THE APPROPRIATE LINE INTO THE NEXT STATE VAR. A 'VECTOR' MUST BE SITTING AT THE APPROPRIATE SITTING IN THE ROM LOCATION THAT IS FORMED BY 'OR'ING THE NEXT STATE AND THE CONTROL LINE. THIS THEN TRANSFERS CONTROL TO THE NEW MICRO ROUTINE. WHEN THE _JMP LINES ARE NOT ACTIVE, THEY ARE PULLED LOW. IN ADDITION TO THE CARRY AND KEYBOARD LINES, THE THREE LOWER BITS OF THE KEYBOARD ARE GATED IN WHEN KEY_JMP IS ACTIVE. VECTORS EXIST FOR ALL POSSIBLE KEY COMBINATIONS, AND FOR FUNCTIONS THEY BRANCH TO SPECIFIC ROUTINES IN THE μ CODE

IC's

FLOW PATTERN: SEE THE PAGE (38) FOR THE ARITHMATIC SYSTEM. (ALSO SEE THE COMMENTED μCODE LISTING). ARITHMATIC IS DONE IN 4 BIT CHUNKS, AND THE CARRY IS AUTOMATICLY ROUTED TO THE NEXT DIGIT. (OVER)

*SEE NOTEBOOK PAGE FOR DESCRIPTION ("THE MICROWORD")

STATE MACHINE, CONT

THE SCHEME FOR STACK/ARITHMATIC CAN BE DESCRIBED AS FOLLOWS: (WHERE BOS IS BOTTOM OF STACK AND TOS IS TOP OF STACK, AND (X) REFERS TO 'THE CONTENTS OF')

CLEAR X: (BOS) ← Ø

KEY: SHIFT (BOS);
(BOS) ← KEY

ENTER: ~~(BOS)~~
(BOS+1) ← (BOS)
INC BOS
IF BOS = TOS THEN
INC TOS

} HAPPENS VIA SPECIAL HARDWARE, NOT STATE MASH.

SUBTRACT: CHS (BOS)
ADD

ADD: DEC BOS
(BOS) ← (BOS) + (BOS+1)
DEC TOS
(TOS) ← (TOS+1)

AGAIN, SEE μ CODE LISTING FOR DETAILS

ROLL*: DEC BOS

* UNNAMED KEY

SOFTWARE TOOLS.

A MICROCODE ASSEMBLER WAS WRITTEN TO CREATE THE μCODE FROM PATTERNS. SEE LISTINGS IN NOTEBOOK FOR FULL DETAILS

SYSTEM USE

THE CALCULATOR WORKS BY TYPING NUMBERS IN, THEN USING THE ENTER (ENT) KEY TO PLACE THEM IN THE STACK. THE CHS KEY CHANGES THE SIGN OF THE X REGISTER & MAY BE USED AT ANY TIME. IF THE NUMBER ENTERED (OR THE RESULT OF THE LAST OPERATION) IS NOT USED AS THE NEXT 'X', THE CLEAR X (CLX) KEY MUST BE USED.*

LOOK! NEW FEATURE → THE UNMARKED KEY PERFORMS A STACK-ROLL DOWN, ALLOWING THE EXAMINATION OF THE OTHER REGISTERS.

CONCLUSION

S*I*G*H. AT 1:30 PM THE CALCULATOR PERFORMS ALL STACK FUNCTIONS AND ADDITION CORRECTLY. THE ADDITION OF A LARGE POSITIVE NUMBER WITH A SMALLER NEGITIVE NUMBER WORKS.... ER... SOMETIMES. THE ROLL-DOWN FEATURE IS COMPLETLY FUNCTIONAL.

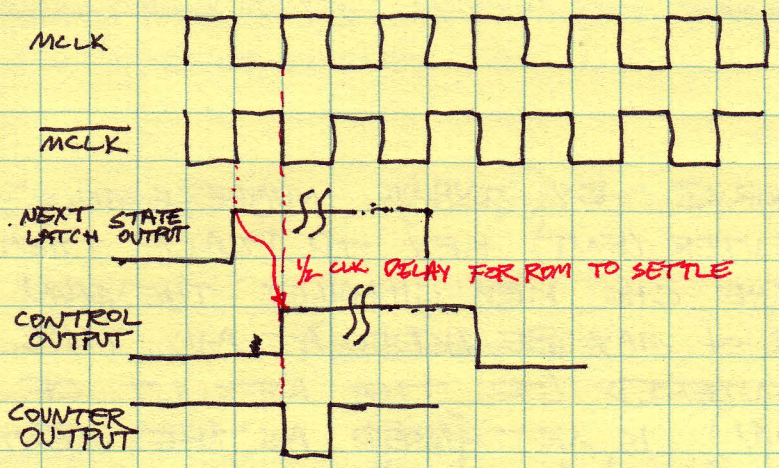
UNFORTUNATLY, TIMING ON THE CARRY-JMP MECHINISM IS HAIRY ENOUGH THAT THE CALCULATOR DOESN'T MAKE THE BRANCHES TO THE PROPER LOCATIONS

(MORE.....)

* SOMEWHAT, ER... "NON-STANDARD"

LAST NOTES....

TIMING (MY EVENTUAL DOWNFUL) ON THE STATE MACHINE WORKED AS FOLLOWS:



Designs are good overall, though there are a couple places which are faulty. In particular, taking outputs from latches through some logic and not involving the clock (can cause glitches occasionally).

Documentation is usually a bit sketchy but certainly picked up with the introduction of the microcode. The information is usually there, just a bit in need of some organization (the binder!) and some "fleshing out".

I would liked to have seen it work, and with a little more time (and a few corrections), I'm sure it would have

Wed	2	(I'll work this one)
Fri	3	
Fri	2	
Design	3.3	
Doc	3.0	
	13.3/15	Bob