Date: 2/13/16

To: CSS 422, Winter 2016

From: Easy Riders

Subject: Team Progress Report 1

We officially formed the Easy Riders group and informally discussed structure and logistics for the project. We will use GitHub for version control and collaborative coding and Google Drive for document content creation. These progress reports will summarize and capture relevant data from those locations.

Status Report:

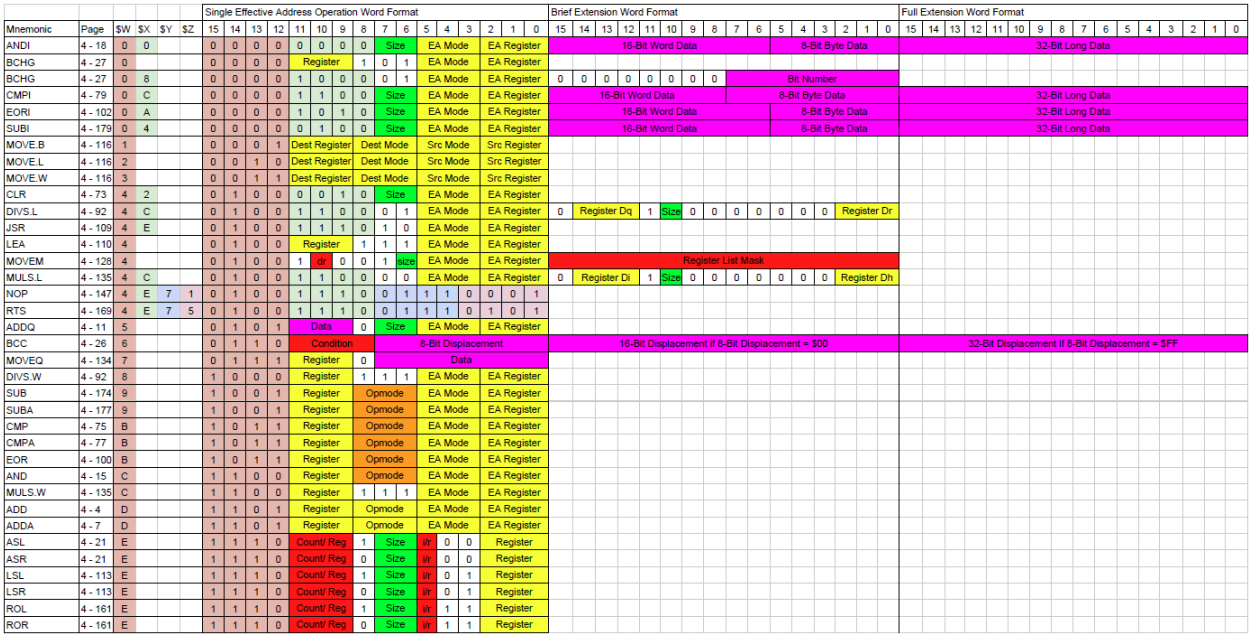
- Finish complete decode for assigned mnemonics and ea modes in Branch List spreadsheet.
    - Notes: Found a definite structure in the operational codes. This will reveal branches necessary in the program structure.
    - Blockers:
    - Assigned: Thomas 2/11
    - Status:
        - Finished organization of all assigned opcodes. td 2/12
        - Finished organization of all assigned ea modes. td 2/12
        - Done. td 2/12
- Create prototype disassembler client hardcoded to convert TestNOP.S68 machine code back into assembler.
    - Notes: Must decode NOP and SIMHALT machine code from address $7000
    - Blockers: Having difficulty printing hex addresses character by character. Using a character table to print each hex character, still trying to figure out how to set up the offsets for the table. rh 2/13 (resolved) rh 2/13
    - Assigned: Ross 2/12
    - Status:
        - Testcode uploaded. td 2/12
        - General program layout established. rh 2/13
        - Hardcoded versions of NOP and SIMHALT routines coded. rh 2/13
        - subroutine for printing hex addresses completed. rh 2/13
- Create flowchart for progress report detailing expected program structure and flow.
    - Notes: Due Saturday at 4:00pm.

- Blockers: Requires prototype to be in progress so we better understand how this will work.
- Assigned:
- Status:
    - Creating branch structure for program layout. td 2/12
    - Root branch layout finished. td 2/13
    - 0 branch layout finished. td 2/13
    - 4 branch layout finished. td 2/13
    - 4E branch layout finished. td 2/13
    - 437 branch layout finished. td 2/13
    - MOVE.B opmode layout finished. td 2/13
    - NOP opmode layout finished. td 2/13
    - DECODE_MODE subroutine layout finished. td 2/13

Deliverables:

Branch List - Instructions

It was necessary to identify patterns in how opcodes are used in the 68000 processor to develop a framework for this project. This visual depiction gave us what we needed to figure out how branching paths would work.



						Single Effective Address Operation Word Format																Brief Extension Word Format																Full Extension Word Format															
Mnemonic	Page	$W	$X	$Y	$Z	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ANDI	4 - 18	0	0			0	0	0	0	0	0	0	0	Size		EA Mode			EA Register			16-Bit Word Data												8-Bit Byte Data								32-Bit Long Data											
BCHG	4 - 27	0				0	0	0	0	Register			1	0	1	EA Mode			EA Register																																	
BCHG	4 - 27	0	8			0	0	0	0	1	0	0	0	0	1	EA Mode			EA Register			0	0	0	0	0	0	0	0	Bit Number																							
CMPI	4 - 79	0	C			0	0	0	0	1	1	0	0	Size		EA Mode			EA Register			16-Bit Word Data												8-Bit Byte Data								32-Bit Long Data											
EORI	4 - 102	0	A			0	0	0	0	1	0	1	0	Size		EA Mode			EA Register			16-Bit Word Data												8-Bit Byte Data								32-Bit Long Data											
SUBI	4 - 179	0	4			0	0	0	0	0	1	0	0	Size		EA Mode			EA Register			16-Bit Word Data												8-Bit Byte Data								32-Bit Long Data											
MOVE.B	4 - 116	1				0	0	0	1	Dest Register				Dest Mode			Src Mode			Src Register																																		
MOVE.L	4 - 116	2				0	0	1	0	Dest Register				Dest Mode			Src Mode			Src Register																																		
MOVE.W	4 - 116	3				0	0	1	1	Dest Register				Dest Mode			Src Mode			Src Register																																		
CLR	4 - 73	4	2			0	1	0	0	0	0	1	0	Size		EA Mode			EA Register																																		
DIVS.L	4 - 92	4	C			0	1	0	0	1	1	0	0	0	1	EA Mode			EA Register	0	Register Dq		1	Size	0	0	0	0	0	0	0	Register Dr																					
JSR	4 - 109	4	E			0	1	0	0	1	1	1	0	1	0	EA Mode			EA Register																																		
LEA	4 - 110	4				0	1	0	0	Register				1	1	1	EA Mode			EA Register																																		
MOVEM	4 - 128	4				0	1	0	0	1	dir	0	0	1	size	EA Mode			EA Register			Register List Mask																															
MULS.L	4 - 135	4	C			0	1	0	0	1	1	0	0	0	0	EA Mode			EA Register	0	Register Dl		1	Size	0	0	0	0	0	0	0	Register Dh																					
NOP	4 - 147	4	E	7	1	0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1																																
RTS	4 - 169	4	E	7	5	0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1																																
ADDQ	4 - 11	5				0	1	0	1	Data			0	Size		EA Mode			EA Register																																		
BCC	4 - 26	6				0	1	1	0	Condition				8-Bit Displacement								16-Bit Displacement if 8-Bit Displacement = $00																																				
MOVEQ	4 - 134	7				0	1	1	1	Register			0	Data																																								
DIVS.W	4 - 92	8				1	0	0	0	Register			1	1	1	EA Mode			EA Register																																		
SUB	4 - 174	9				1	0	0	1	Register			Opmode			EA Mode			EA Register																																		
SUBA	4 - 177	9				1	0	0	1	Register			Opmode			EA Mode			EA Register																																		
CMP	4 - 75	B				1	0	1	1	Register			Opmode			EA Mode			EA Register																																		
CMPA	4 - 77	B				1	0	1	1	Register			Opmode			EA Mode			EA Register																																		
EOR	4 - 100	B				1	0	1	1	Register			Opmode			EA Mode			EA Register																																		
AND	4 - 15	C				1	1	0	0	Register			Opmode			EA Mode			EA Register																																		
MULS.W	4 - 135	C				1	1	0	0	Register			1	1	1	EA Mode			EA Register																																		
ADD	4 - 4	D				1	1	0	1	Register			Opmode			EA Mode			EA Register																																		
ADDA	4 - 7	D				1	1	0	1	Register			Opmode			EA Mode			EA Register																																		
ASL	4 - 21	E				1	1	1	0	Count/ Reg			1	Size		ir	0	0	Register																																		
ASR	4 - 21	E				1	1	1	0	Count/ Reg			0	Size		ir	0	0	Register																																		
LSL	4 - 113	E				1	1	1	0	Count/ Reg			1	Size		ir	0	1	Register																																		
LSR	4 - 113	E				1	1	1	0	Count/ Reg			0	Size		ir	0	1	Register																																		
ROL	4 - 161	E				1	1	1	0	Count/ Reg			1	Size		ir	1	1	Register																																		
ROR	4 - 161	E				1	1	1	0	Count/ Reg			0	Size		ir	1	1	Register																																		

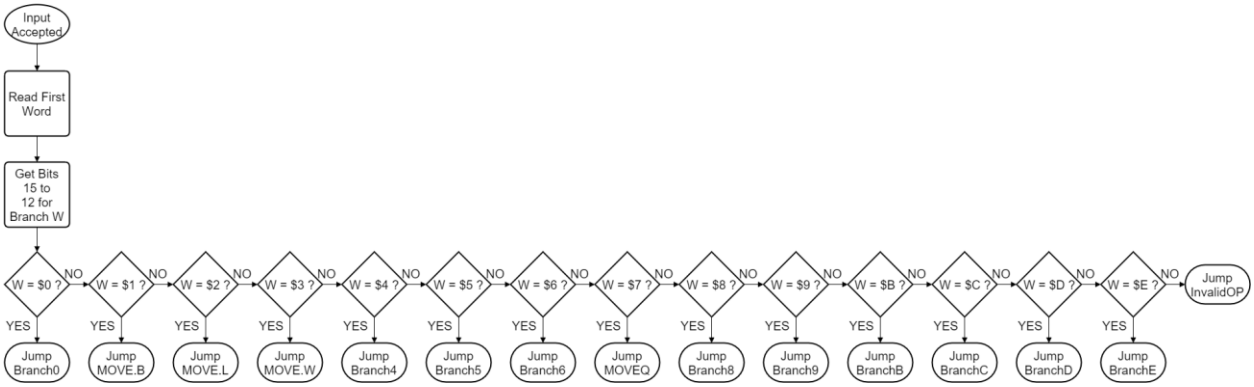
Branch List - EA Modes

To reduce code duplication, a similar strategy was used with addressing modes. Again, a common syntax was found in the structure of modes and registers.

Addressing Modes		Syntax	Mode	Reg
Register Direct	Data	Dn	0 0 0	0 0 0 -> 1 1 1
	Address	An	0 0 1	0 0 0 -> 1 1 1
Register Indirect	Address	(An)	0 1 0	0 0 0 -> 1 1 1
	Addr with Post Inc	(An)+	0 1 1	0 0 0 -> 1 1 1
	Addr with Post De	-(An)	1 0 0	0 0 0 -> 1 1 1
Absolute Data Addr	Short	(xxx).W	1 1 1	0 0 0
	Long	(xxx).L	1 1 1	0 0 1
Immediate		#<xxx>	1 1 1	1 0 0

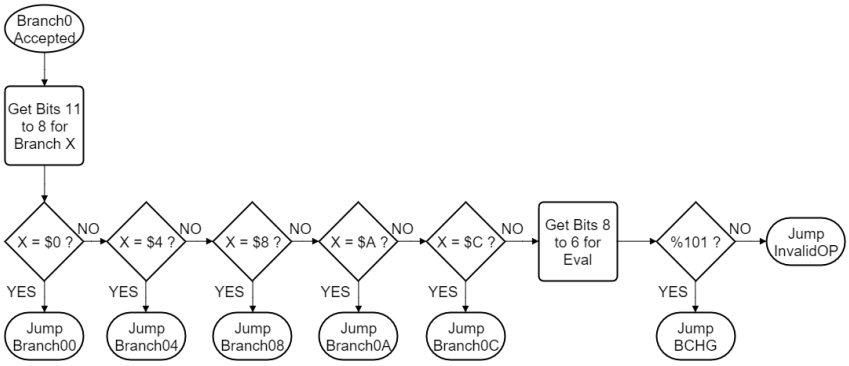
Root Branch Layout

After input, the first four bits are analyzed to determine which command branch to follow. Based off of this, some specific commands are found (MOVE) but most opcodes require further branching.



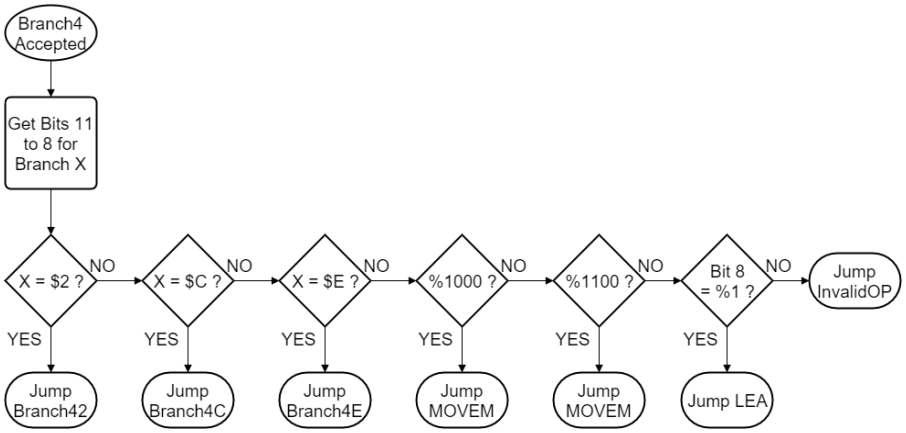
0 Branch Layout

The $0 branch was explored in an attempt to verify that this methodology was going to produce successful results.



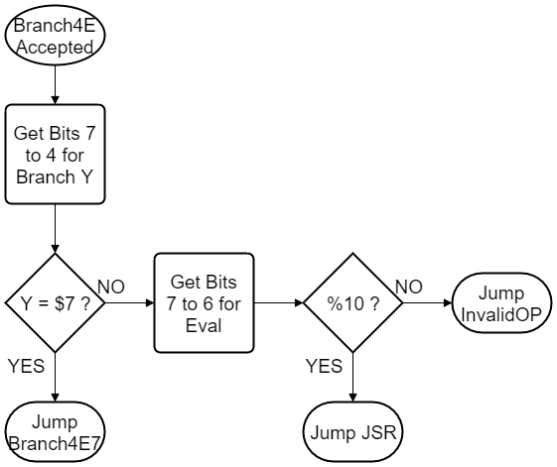
4 Branch Layout

The $4 branch was also followed to explore a full branch path to the NOP opcode.



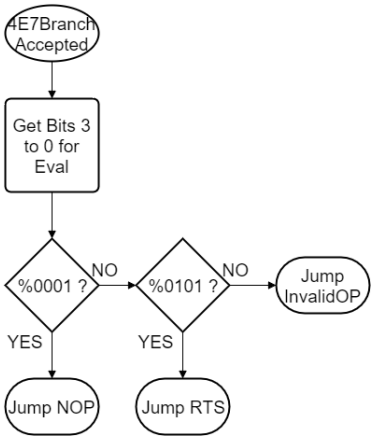
4E Branch Layout

The $4E branch shows a second level in the tree structure.



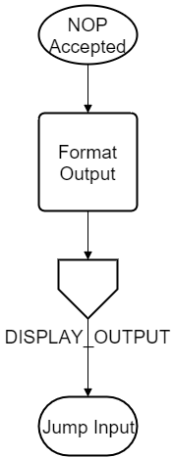
4E7 Branch Layout

The $4E7 branch terminates with two valid opcodes: NOP and RTS



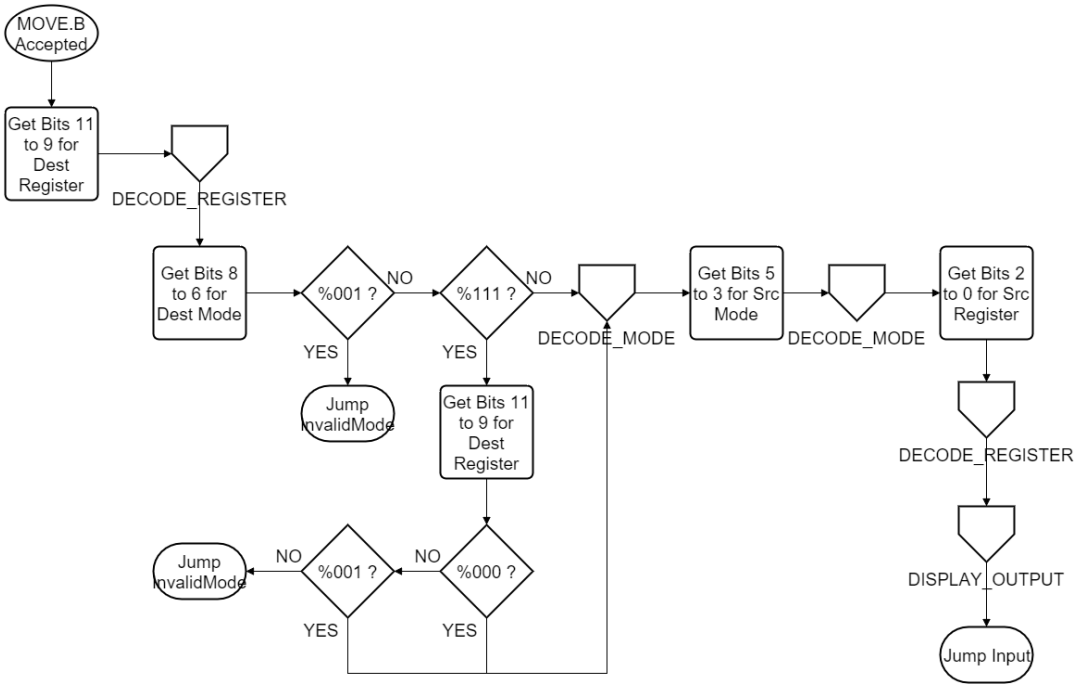
NOP Opcode Layout

The NOP opcode is explored as it is part of the prototype and introduces the concept of subroutines (DISPLAY_OUTPUT).



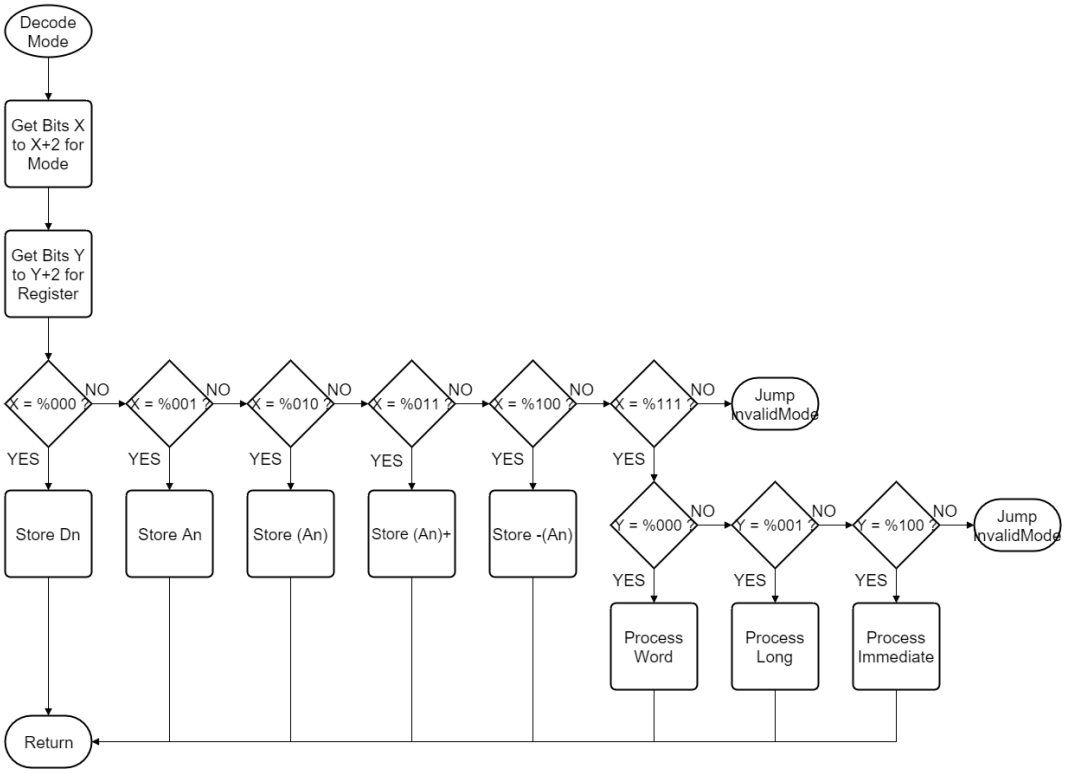
MOVE.B Opcode Layout

The MOVE.B opcode is more complicated and is also directly explored to verify that this methodology will work for our project. Logic for mode validation is detailed out in the flow chart layout.



DECODE_MODE Subroutine Layout

Commonly used code can be blocked into subroutines to prevent excessive duplication. While these don't specifically show implementation, it does depict general flow to arrive at the correct outcome.



Tentative Register Map

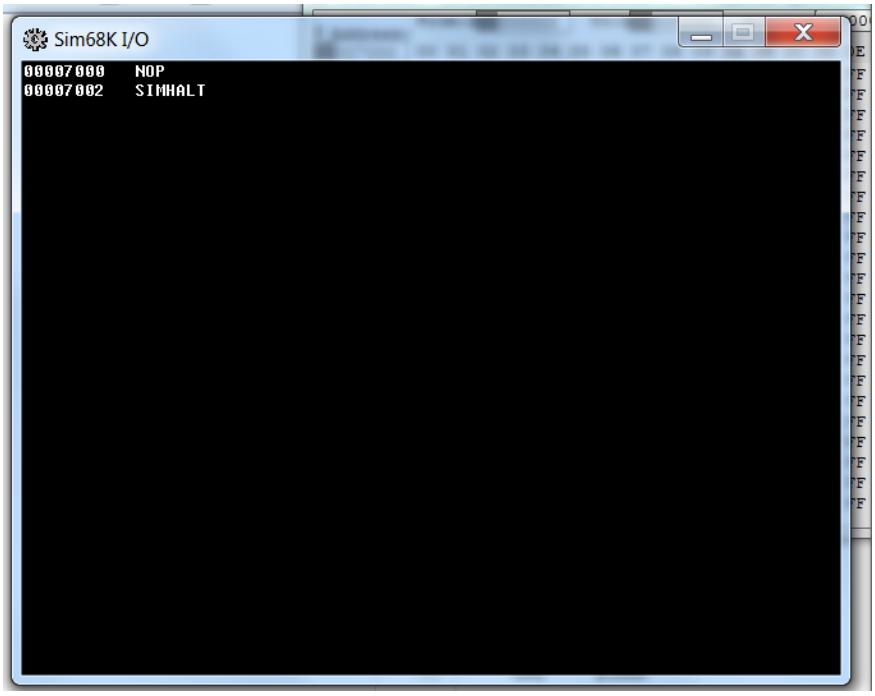
As the scope of this program begins to reveal itself, it's becoming clear we need to standardize how we're storing data in memory so we're not constantly defining new rulesets for every block of code. This is a first attempt which is likely subject to change when we start coding.

- A0:
- A1: Used for TRAP #15 (user input and output strings)
- A2:
- A3:
- A4:
- A5:
- A6:
- D0: Used for TRAP #15 (store trap task)
- D1: Used for TRAP #15 (data from user inputs)
- D2: Operate on bits 15 - 12 (opcode)
- D3: Operate on bits 11- 9 (register)
- D4: Operate on bits 8 - 6 (dest mode, opmode, size)

- D5: Operate on bits 5 - 3 (ea mode)
- D6: Operate on bits 2 - 0 (ea register)
- D7:

Prototype Output

The hex addresses are generated by isolating each character in the address and outputting a corresponding character from a hex character table (0-F). The OPcodes are generated by looking at the 4 most significant bits of each word in memory and using a jump table to move to the corresponding subroutine. Since the test program only has two opcodes there is very little logic required in each subroutine.



Problems:

The biggest hurdle at the moment is dealing with the unknowns related to the project and its structure. The logical flow we can work through, but many decisions are based on guesses without knowing exactly how implementation is going to change things. The second challenge is developing a routine for reporting and adding deliverables. Neither situation is currently blocking or slowing progress.

Work Scheduled:

Assigned tasks from the 2/13 sprint were all completed as scheduled. For the next sprint, these are the highest priority tasks:

- Need full project timeline
- Need to write input branch layout
- Need to finish defining the register mapping
- Transfer completed layouts to project source code

- Continue other branch, opcode, and subroutine layouts

Self Evaluation:
As of right now, we are ahead of schedule with a lot accomplished in two days. The project seems to be on track.