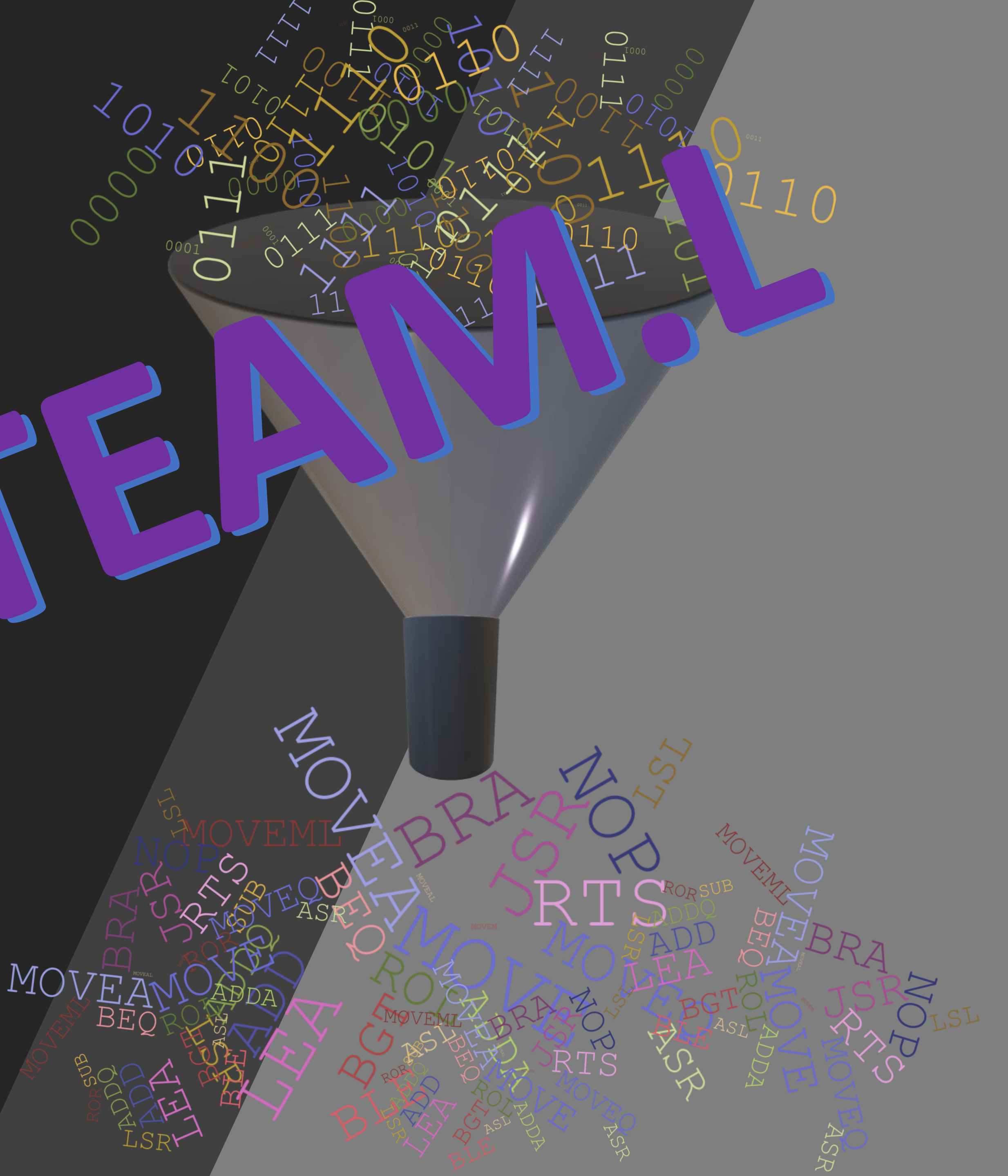
Disassembler

Benjamin Lin | Hamza Siddiqui
Keziah May | Maxwell Wenger

Disassembler

DREAMTEAM.L

Benjamin Lin [illegible]amz[illegible] [illegible]ddi[illegible]ui
Keziah M[illegible] | [illegible]axwel[illegible]e[illegible]



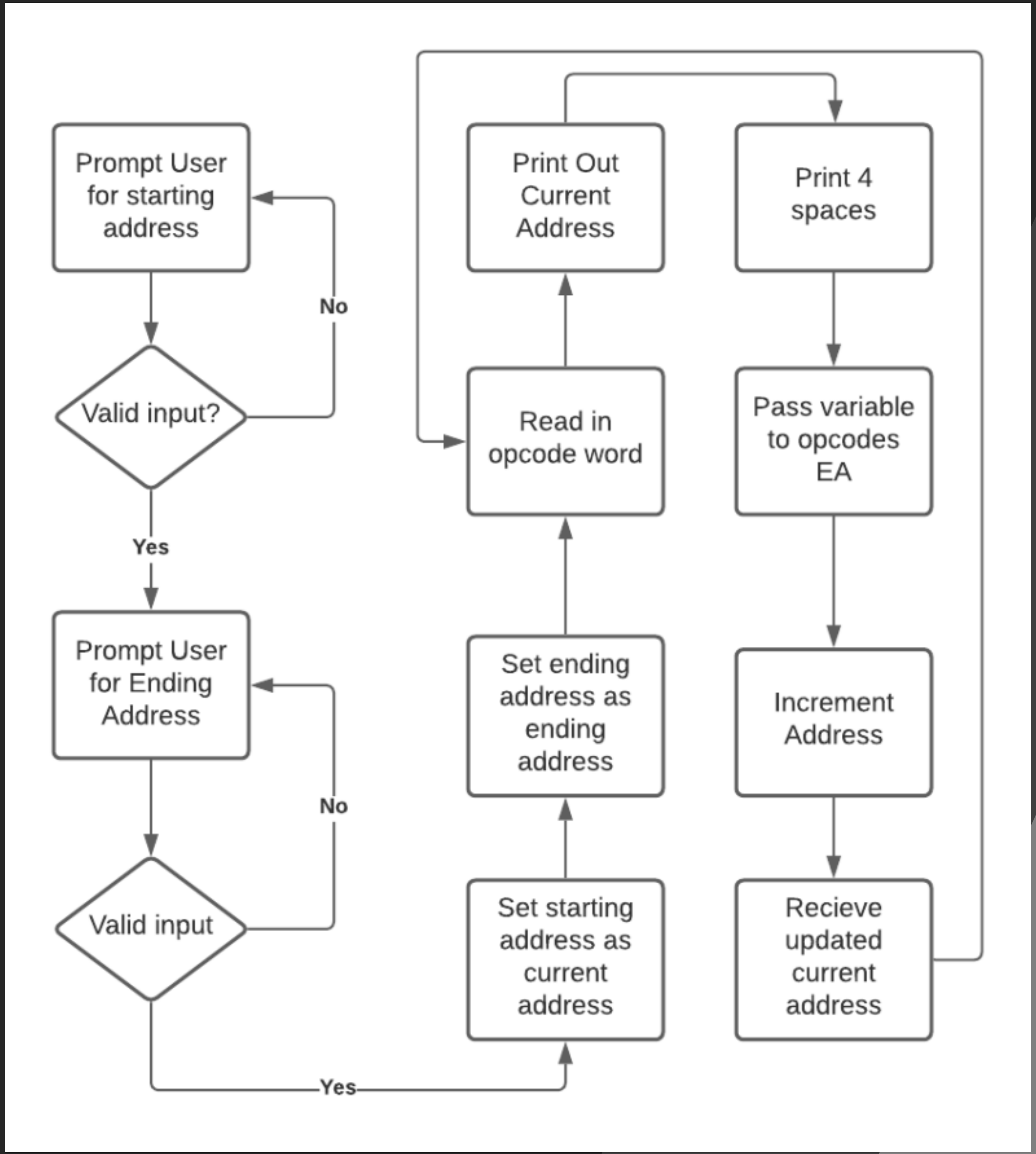
Overview

- IO
- OPCODES
- EA
- Integration
- Issues/Simulator Bugs
- Conclusions

Input/Output

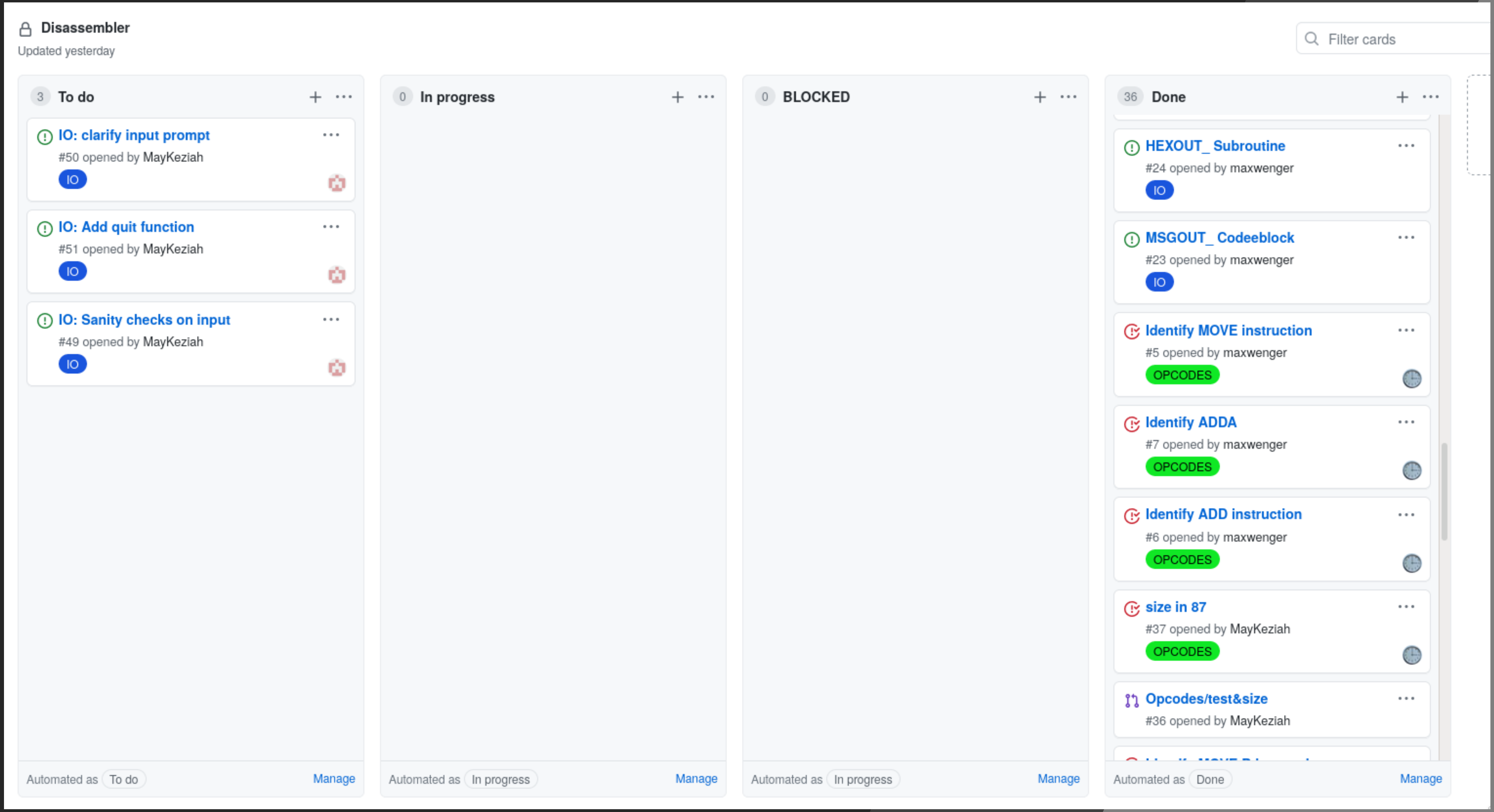
Comprehensive list of items completed and items to do.

I/O



Prompt User for starting address
Valid input?
No
Yes
Prompt User for Ending Address
Valid input
No
Yes
Set starting address as current address
Set ending address as ending address
Read in opcode word
Print Out Current Address
Print 4 spaces
Pass variable to opcodes EA
Increment Address
Recieve updated current address

I/O



Disassembler
Updated yesterday

Filter cards

3 To do

IO: clarify input prompt
#50 opened by MayKeziah
IO

IO: Add quit function
#51 opened by MayKeziah
IO

IO: Sanity checks on input
#49 opened by MayKeziah
IO

Automated as To do Manage

0 In progress

Automated as In progress Manage

0 BLOCKED

Automated as In progress Manage

36 Done

HEXOUT_ Subroutine
#24 opened by maxwenger
IO

MSGOUT_ Codeblock
#23 opened by maxwenger
IO

Identify MOVE instruction
#5 opened by maxwenger
OPCODES

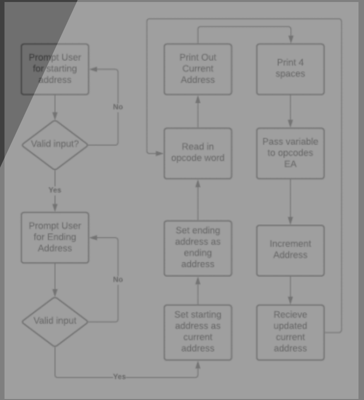
Identify ADDA
#7 opened by maxwenger
OPCODES

Identify ADD instruction
#6 opened by maxwenger
OPCODES

size in 87
#37 opened by MayKeziah
OPCODES

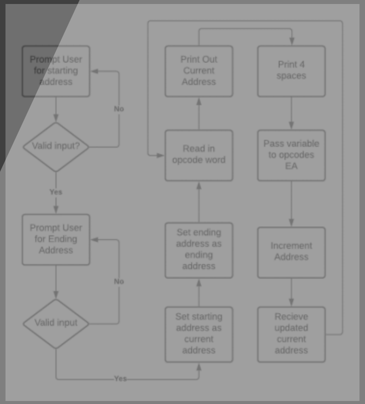
Opcodes/test&size
#36 opened by MayKeziah

Automated as Done Manage


Prompt User for starting address
Valid input?
No
Yes
Prompt User for Ending Address
Valid input
No
Yes
Set starting address as current address
Set ending address as ending address
Read in opcode word
Print Out Current Address
Print 4 spaces
Pass variable to opcodes EA
Increment Address
Recieve updated current address

I/O

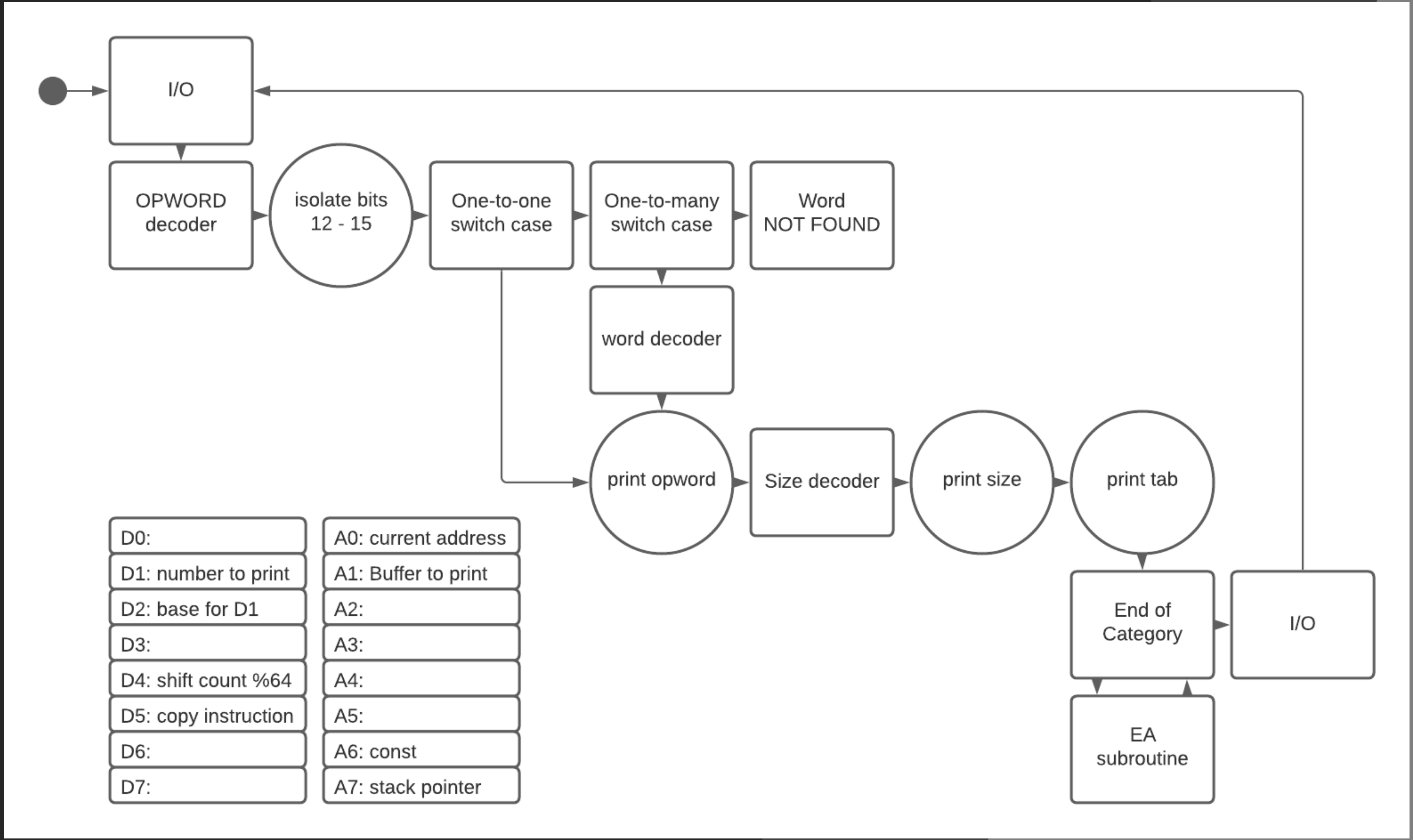
Done
1. Prompt for start address
2. Prompt for end address
3. Read instruction in memory to register
4. Display current address and tab
5. Call OPCODE module
6. Increment address pointer
7. Loop through memory to end address
8. Loop prompt for input


Prompt User for starting address
Valid input?
No
Yes
Prompt User for Ending Address
Valid input
No
Yes
Set starting address as current address
Set ending address as ending address
Read in opcode word
Print Out Current Address
Print 4 spaces
Pass variable to opcodes EA
Increment Address
Recieve updated current address

OPCODES

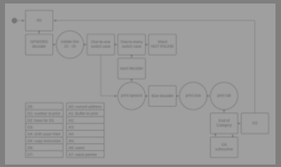
Comprehensive list of items completed.

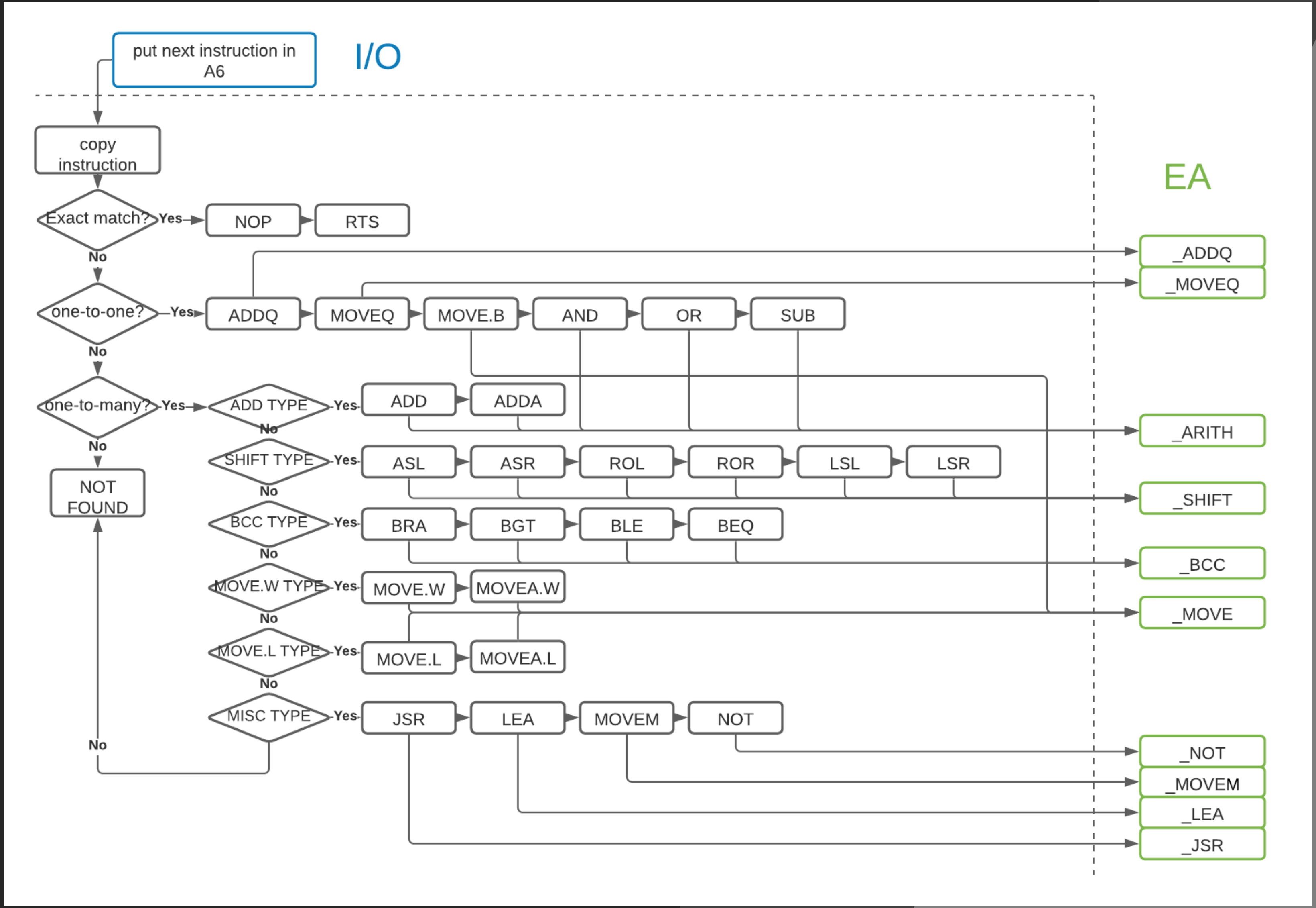
OPCODES



D0:	A0: current address
D1: number to print	A1: Buffer to print
D2: base for D1	A2:
D3:	A3:
D4: shift count %64	A4:
D5: copy instruction	A5:
D6:	A6: const
D7:	A7: stack pointer

OPCODES

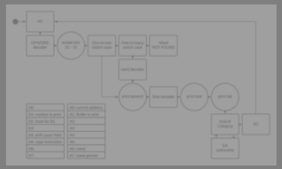
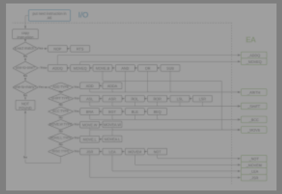




put next instruction in A6
I/O
copy instruction
Exact match?
Yes
No
NOP
RTS
one-to-one?
Yes
No
ADDQ
MOVEQ
MOVE.B
AND
OR
SUB
one-to-many?
Yes
No
NOT FOUND
ADD TYPE
Yes
No
ADD
ADDA
SHIFT TYPE
Yes
No
ASL
ASR
ROL
ROR
LSL
LSR
BCC TYPE
Yes
No
BRA
BGT
BLE
BEQ
MOVE.W TYPE
Yes
No
MOVE.W
MOVEA.W
MOVE.L TYPE
Yes
No
MOVE.L
MOVEA.L
MISC TYPE
Yes
No
JSR
LEA
MOVEM
NOT
EA
_ADDQ
_MOVEQ
_ARITH
_SHIFT
_BCC
_MOVE
_NOT
_MOVEM
_LEA
_JSR

OPCODES

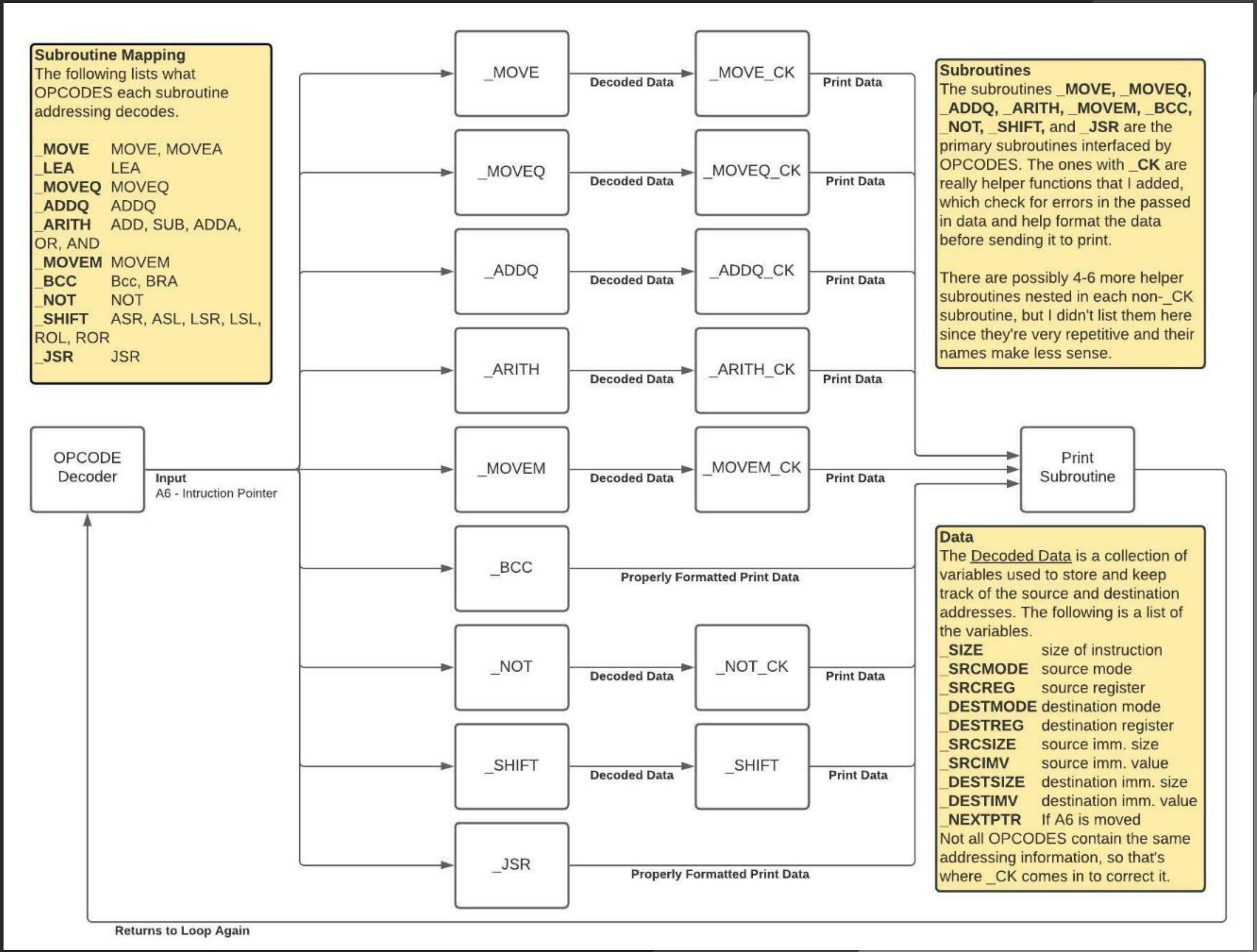
Done: decode & display opword, size, and tab
Exact Match
RTS
NOP
One to One
ADDQ
AND
OR
SUB
MOVEQ
MOVE.B
One to Many
ADD, ADDA
ASL, ASR, LSL, LSR, ROL, ROR
Bcc (BGT, BLE, BEQ), BRA
JSR, LEA, MOVEM, NOT
MOVE.W, MOVEA.W
MOVE.L, MOVEA.L

EA

Comprehensive list of items completed.

EA



Subroutine Mapping
The following lists what OPCODES each subroutine addressing decodes.

_MOVE	MOVE, MOVEA
_LEA	LEA
_MOVEQ	MOVEQ
_ADDQ	ADDQ
_ARITH	ADD, SUB, ADDA, OR, AND
_MOVEM	MOVEM
_BCC	Bcc, BRA
_NOT	NOT
_SHIFT	ASR, ASL, LSR, LSL, ROL, ROR
_JSR	JSR

Subroutines
The subroutines **_MOVE, _MOVEQ, _ADDQ, _ARITH, _MOVEM, _BCC, _NOT, _SHIFT,** and **_JSR** are the primary subroutines interfaced by OPCODES. The ones with **_CK** are really helper functions that I added, which check for errors in the passed in data and help format the data before sending it to print.

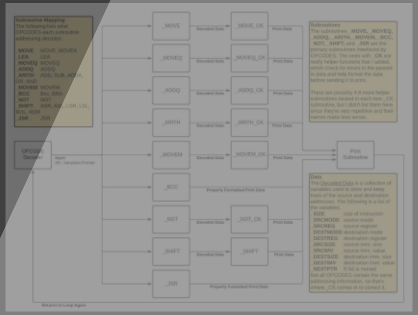
There are possibly 4-6 more helper subroutines nested in each non-_CK subroutine, but I didn't list them here since they're very repetitive and their names make less sense.

Data
The Decoded Data is a collection of variables used to store and keep track of the source and destination addresses. The following is a list of the variables.

_SIZE	size of instruction
_SRCMODE	source mode
_SRCREG	source register
_DESTMODE	destination mode
_DESTREG	destination register
_SRCSIZE	source imm. size
_SRCIMV	source imm. value
_DESTSIZE	destination imm. size
_DESTIMV	destination imm. value
_NEXTPTR	If A6 is moved

Not all OPCODES contain the same addressing information, so that's where _CK comes in to correct it.

EA

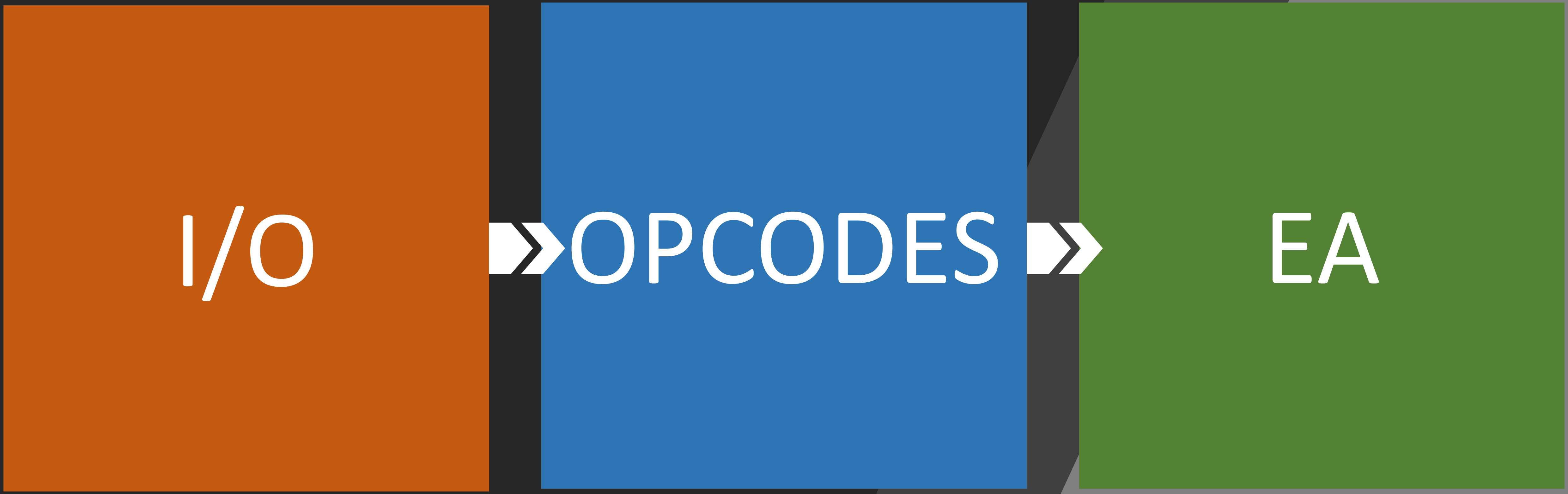


Done
1. Implement provided interface subroutines for OPCODES
2. Create error handling for each subroutine
3. Create printing function for each subroutine
4. Decode all required Effective Addressing Modes: • Data Register Direct • Address Register Direct • Address Register Indirect • Immediate Data • Address Register Indirect with Post incrementing • Address Register Indirect with Pre decrementing • Absolute Long Address • Absolute Word Address

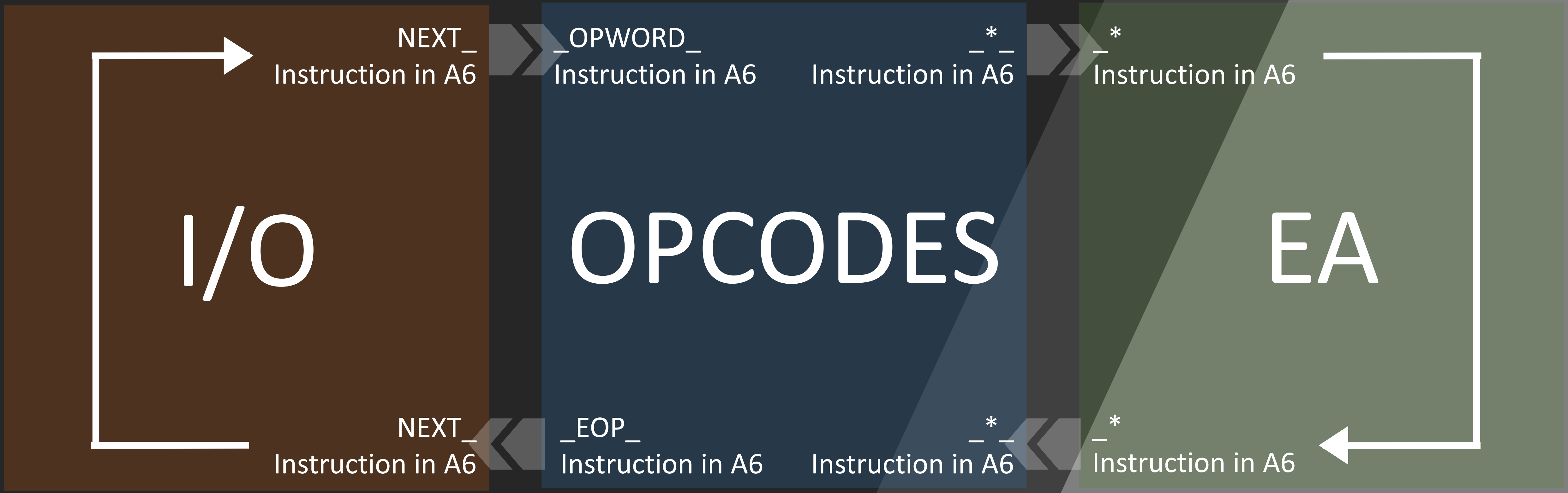
Integration

Comprehensive list of items completed.

Integration



Integration



Integration

IO/OPCODE Interface	
IO Required	OPCODE Provided
NEXT_	_OPWORD_
OPCODE/EA Interface: All return	
OPCODE Required	EA Provided
EMOVE	_MOVE
LEA	_LEA
MOVEQ	_MOVEQ
RTEOPAQ	_ADDQ
RTEOP8	_ARITH
EMOVEM	_MOVEM
EBCC	_BCC
ENOT	_NOT
EOFSHIFT	_SHIFT
JSR	_JSR
OPCODE/IO Interface	
OPCODE Required	IO Provided
EOP	NEXT_

Issues

What we struggled with and how we responded.

Issues

Along the way, these concepts slowed us down:

- Remembering to clear registers
- Incrementing/decrementing the memory pointer
- Integration miscommunication
  - Naming conventions
  - Planning register usage
  - Agreeing on which function does what
- Handling errors
- Standardizing error handling
- Using wrong size
- Tracking test cases before integration

Issues

Error Handling Display Standards:

- **OPCODES**

  Invalid OPCODE:

  ```
  XXXX    DATA       YYYY
  ^         ^          ^
  Memory  Error      Hex instruction unable to decode
  address Display    located at XXXX in Memory
  ```

  Invalid size:

  ```
  XXXX    ADDQ.X     OP1,OP2
  ^            ^     ^
  Memory  Error      EA's display for
  address size       this instruction
          display
  ```

- **EA**

  Invalid operand

  ```
  1A00    MOVE.W     <invalid EA mode>, D1
  1A02    MOVE.L     (A4)+, <invalid EA mode>
  1A02    MOVE.B      <invalid EA mode>, <invalid EA mode>
  ```

Issues

Error Handling Display Standards:

- **OPCODES**

```
Invalid OPCODE:
XXXX    DATA        YYYY
^         ^           ^
Memory  Error      Hex instruction unable to decode
address Display    located at XXXX in Memory

Invalid size:
XXXX    ADDQ.X      OP1,OP2
^            ^        ^
Memory    Error      EA's display for
address   size       this instruction
          display
```

- **EA**

```
Invalid operand
1A00    MOVE.W      <invalid EA mode>, D1
1A02    MOVE.L      (A4)+, <invalid EA mode>
1A02    MOVE.B       <invalid EA mode>, <invalid EA mode>
```

Not all opcodes follow this pattern.

Some one-to-many result in an invalid opword instead of an invalid size.

Conclusions

Advice for future CSS432 students.

Conclusions

- Start early
- Plan interfaces and naming conventions
- Dedicate registers to specific tasks
- Create a list of test cases for each OPCODE and EA
- Find patterns to categorize OPCODES and EA
- Test and Commit to your branch often
- Meet often to confirm your integration plans
- Integrate early
- Do extensive integration testing
- Test error handling cases
- Try not to cry