Software Design Document (SDD)
Assignment #2 DXE Disassembler for XE computer
CS530, Spring 2020

Team:

Nathan Azoulay, cssc1915,
Angelo Cabading, cssc1952,
Michael Hoang, cssc1954,
Destyni Ta, cssc1953

Overview & Goals:

Pseudo Code:

```
Check for file input errors
    Display error message if incorrect -> break out
Open file
Initialize our data to read in symbol file
Read symbol file, store as vector (might change)
Read object file, create output files with code
Check over symbol file addresses
    Write RESW or RESB if a symbol address is not listed
```

Goal (by 3/30):

- Check for file input errors (NA)
- Initialize data to read (NA)
- Read symbol file (Determine what structure to store in, vector, map etc.) **- Nathan, Destyni,Michael, Angelo**
- Start reading object file **-Nathan**
    - Figure out how to execute our logic in disassembling the code
    - Determine how to break down a modification record

Goal (by 4/10):

- Finish reading object file and disassembling the code -
- Recheck symbol file addresses for allocated memory -
- Debug program (gives us about 10 days) to get program fully functioning
- Test program with various other symbol tables and object files
- Read over comments, make sure everything is clear and understandable
- Determine if there are any cases or instructions that are not working, if so why?

Project Description:

This project will contain an XE disassembler program that opens an object file and its accompanying symbol file, labeled <filename>.obj and <filename>.sym respectively. Upon running the disassembler executable program named ‘dxe’, an XE source file, <filename>.sic, and a corresponding XE listing file, <filename>.lis, will be generated by the program upon disassembling the object code. Additionally, <filename>.sym will then also contain the SYMTAB and LITTAB that was generated during the disassembly process. The disassembler will then use “filename” for the name of the source file it generates. If neither the <filename>.obj and <filename>.sym are present then the xed program shall exit.

Plan of Action and Milestones:

3/9

- Completed file checking, reading file with file pointer (NA)
- Created struc with Opcode Table and a struct with library of registers (NA)
- Planned logic on whiteboard (ALL)

3/13

- Begin coding function to read in symbol table (NA)
- Start building overview on how to read in obj table (flag variables etc) (NA)
- Functions to convert hexadecimal to binary

3/20

- Code to disassemble object file (NA)
    - Begin reading flags and executing logic described in system design (NA)
- Start creating output files and storing results (NA)

4/10

- Program should be functioning with minor bugs
- Ready for debugging and test phase
- Read/Add comments (NA)
- Turn in project by April 20

4/20

- Project is not fully functioning or completed.
- Read symbol table and store in our data structures (vectors) works, tested fully (Nathan)
- Reading object file is not complete, I was able to finish the cases for header record, and part of text records (Nathan)
- Able to set the n and i flags correctly, along with proper addressing formats (Nathan)

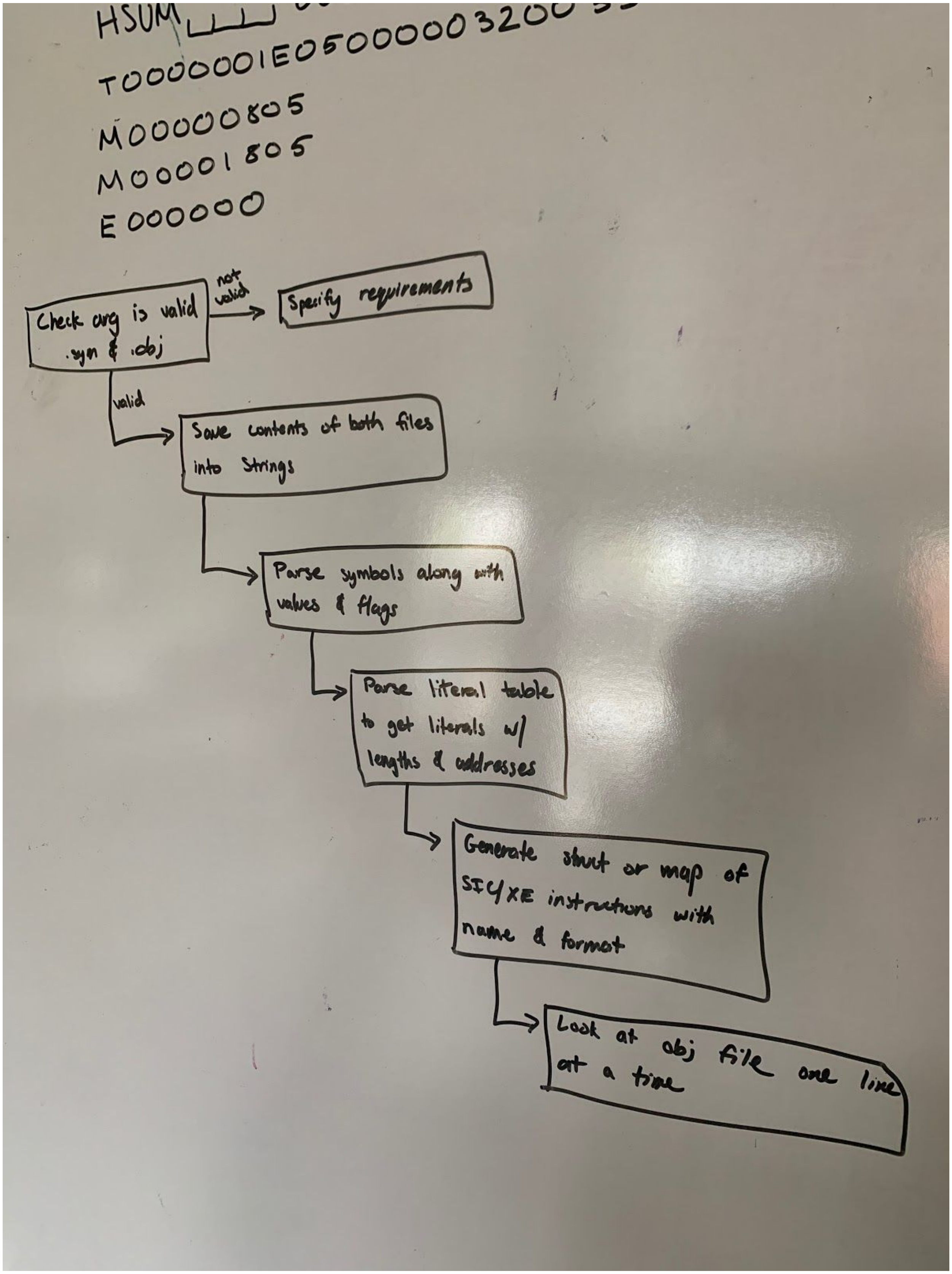
Requirements:

- xed.cpp (Disassembler file)
- May have other cpp files present for conversions or certain functions we may need
- README file
- Makefile
- filename.sym
- filename.obj
- filename.lis
- filename.sic

Development Environment:
Visual Studio Code

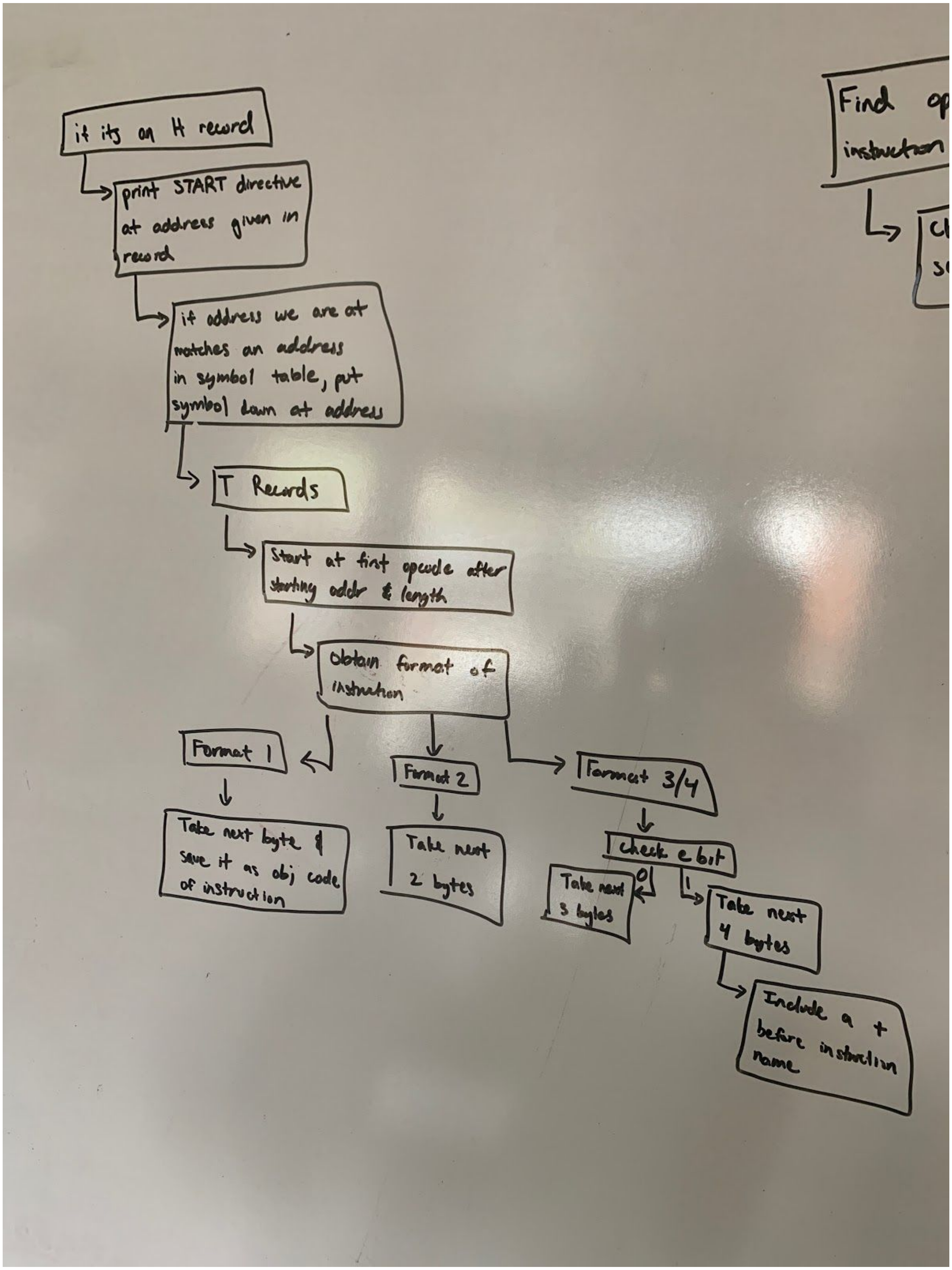
Run/Test Environment:
Edoras

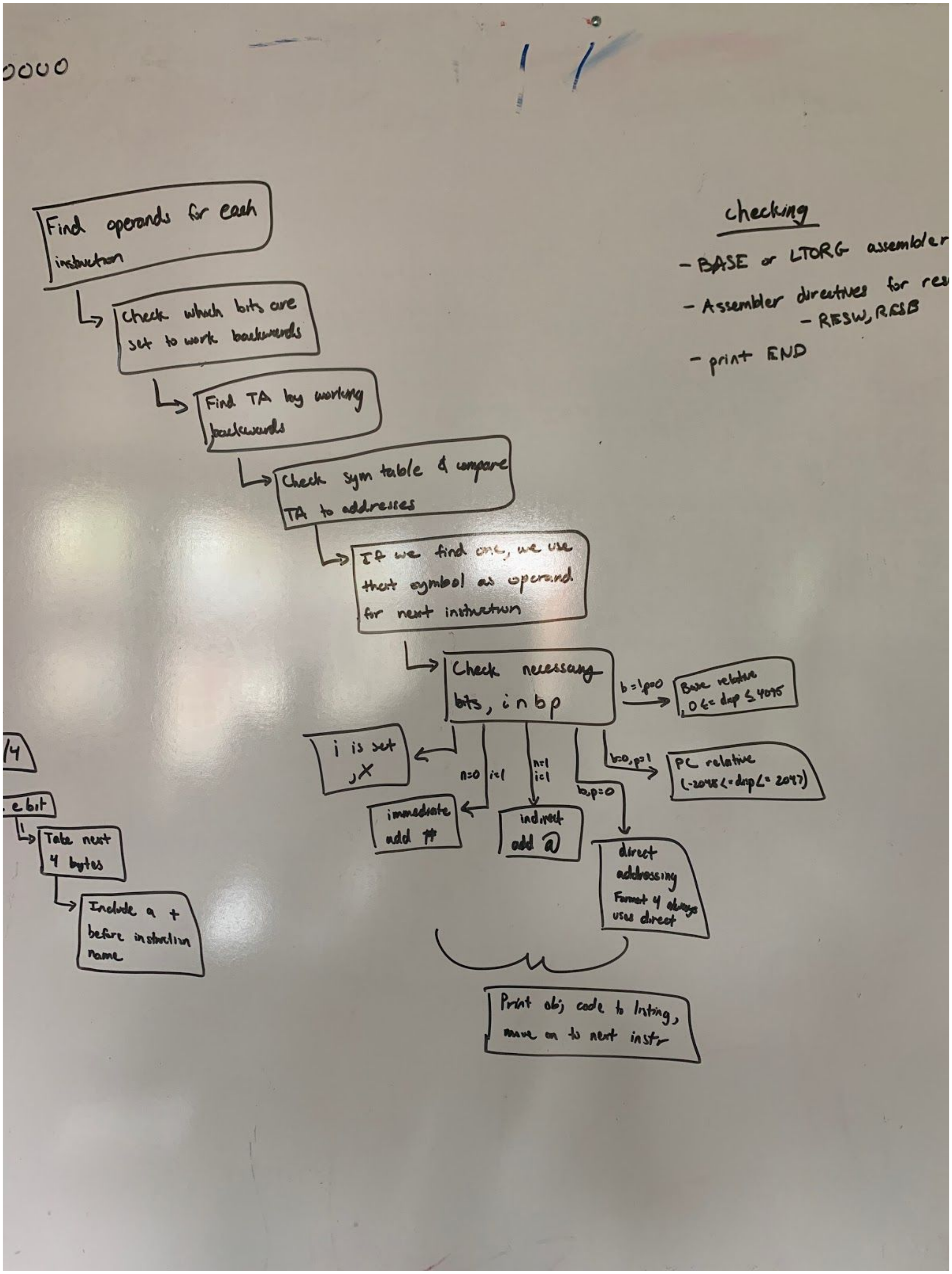
System Design/Specification:



```
HSUM
T000001E0500000320
M00000805
M00001805
E000000
```

Check arg is valid .sym & .obj
not valid
Specify requirements
valid
Save contents of both files into Strings
Parse symbols along with values & flags
Parse literal table to get literals w/ lengths & addresses
Generate struct or map of SIC/XE instructions with name & format
Look at obj file one line at a time


if its an H record
print START directive at address given in record
if address we are at matches an address in symbol table, put symbol down at address
T Records
Start at first opcode after starting addr & length
obtain format of instruction
Format 1
Take next byte & save it as obj code of instruction
Format 2
Take next 2 bytes
Format 3/4
check e bit
0
Take next 3 bytes
1
Take next 4 bytes
Include a + before instruction name
Find op
instruction
Ch
s


0000
Find operands for each instruction
Check which bits are set to work backwards
Find TA by working backwards
Check sym table & compare TA to addresses
If we find one, we use that symbol as operand for next instruction
Check necessary bits, i n b p
b=1 p=0
Base relative 0 <= disp ≤ 4095
b=0, p=1
PC relative (-2048 <= disp <= 2047)
i is set
,X
n=0 i=1
immediate add #
n=1 i=1
indirect add @
b,p=0
direct addressing
Format 4 always uses direct
Print obj code to listing, move on to next instr
/4
e bit
1
Take next 4 bytes
Include a + before instruction name

checking

- BASE or LTORG assembler
- Assembler directives for res
  - RESW, RESB
- print END