Natalia Gilbertson, Thomas Kercheval, Saam Amiri
3/16/2018
Disassembler: Program Specification

We would like to enter the extra OP code contest.

What Our Program Does
Our disassembler is a program that takes as input into the program memory an executable .S68 file. The program iterates over the input memory to identify 68K assembly instructions by their opcodes and effective-addressing modes. The program will print one page of disassembled code to the output for each ENTER the user presses. Any lines outputted as XXXXXXXX DATA $YYYY represent any unsupported opcodes/instructions that were not disassembled into assembly (XXXXXXXX representing the address where the invalid/unsupported instruction was found, $YYYY representing the instruction word that could not be disassembled).

These are the opcodes that are supported by our disassembler:

ORI	BLT
BCLR	DIVS
CMPI	OR
BCLR	SUB
MOVEA	CMP
MOVE	EOR
NEG	MULS
RTS	ADD
JSR	ADDA
MOVEM	ASR
LEA	LSR
SUBQ	ROR
BRA	ASL
BCS	LSL
BVC	ROL
BGE	

These are the extra (not required) opcodes supported by our disassembler:

ANDI	BPL
SUBI	BMI
EORI	BGT
ADDI	BLE
BTST	CLR
BHI	TRAP #15
BLS	AND
BCC	MULU
BNE	CMPA
BEQ	SUBA
BVS	NOP
ADDQ	

These are the EA modes that are supported by our disassembler:

- Data Register Direct: Dn
- Address Register Direct: An
- Address Register Indirect: (An)
- Address Register Indirect with Post incrementing: (A0)+
- Address Register Indirect with Pre decrementing: -(SP)
- Immediate Data: #
- Absolute Long Address: (xxx).L
- Absolute Word Address: (xxx).W

How To Use Our Disassembler

1. Open Easy68K and open the file src/io_NataliaGilbertson.x68.
2. Press the green ‘run’ arrow in Easy68K and then press ‘Execute’.
3. Now that the I/O code is open in the execution window, go to File…, Open Data…, and select the .S68 file that you want to disassemble.
    Note: Make sure that the file you want to load into memory for disassembly is not ORG’d at a place where it will overwrite our program in memory, or else we cannot disassemble your data.
4. Now press the green ‘run’ arrow in the execution window.
5. The execution will stop and prompt the user to enter the starting address of the data-to-be-disassembled in memory. Type in the starting address and press ENTER.
    Note: If an invalid address was inputted, the program will print an error message and ask the user to enter a valid starting address again.
6. The execution will stop and prompt the user to enter the ending address of the data-to-be-disassembled in memory. Type in the ending address and press ENTER.
    Note: If an invalid address was inputted, the program will print an error message and ask the user to re-enter a valid addresses again.
7. Next, the execution will stop and prompt the user to press ENTER for the next page of disassembled output, waiting for the user to press ENTER. Press ENTER when you are ready to see the first page of code.
8. For each remaining page of input to be disassembled, execution will stop and wait for the user to press ENTER. Press ENTER when you are ready to see the next page of code.
9. When the last page of output is printed to the console, the disassembler will print an “~End of File~” message to the console. You may now close the execution.

Sexy8K’s Code Standards

- Use labels unique to your module.
    - Don’t use basic labels such as ‘loop’, ‘end’, ‘exit’
- Comment all lines that are not trivial.
    - No need to comment lines such as incrementing a register or clearing a register.
- Give descriptions for chunks of related code.
    - For chunks of code that are a few lines long that do a specific task, give an general description/comment for what that chunk does.
- Define variables at the bottom of the source file.
- Only use A7 for the stack (do not touch it otherwise).

- Use a standard git workflow for version control.
- Maintaining the stack.
- Determine upfront which registers are reserved for parameters passed between modules (refer to API).
- Use functions as much as possible to modularize code.
- Minimize redundant code (follow D.R.Y./W.E.T. principle).
- Use stars for comments, no semicolons.
- No using tabs, use spaces (don’t mix the two either...seriously).
- Notify the team over discord when you push changes to the repo.
- Flowchart before coding.
- Test your code before pushing to master.