Mmn 14 - Summary

1. Handling in
    a. *.c, *.h files
    b. Compile & linked(exe) file.
    c. makefile.
    d. Assembly inputs & outputs files/screenshots of errors.

2. Hardware
    a. 8 registers - r0 - r7, each register is 14 bits.
    b. PSW(program status word) register - flags for CPU.
    c. RAM - 256 cells (0 - 255), each cell is 14 bits. A cell is also known by the name "word."
    d. Numbers can be only integers using 2's complement for negatives.
    e. Characters represented by their ASCII code.

3. Word
    a. Bits values:

13	12	11	10	9	8	7	6	5	4	3	2	1	0
par1		par2		opcode				srcAdd		destAdd		E,R,A	

    b. 0-1: E,R,A
        i. **A = Absolute = 00**, content isn't dependent where the machine code will be loaded. Ex. A number.
        ii. **E = External = 01**, content depends on an external symbol. Ex. A label that is defined in a different file from the word.
        iii. **R = Relocatable = 10**, content is dependent where the machine code will be loaded. Ex. A label is defined in the same file as the word.
    c. 2-3: destination operand's addressing mode.
        i. **00 = instant number**. Ex. #4.
        ii. **01 = label**. Ex. HELLO.
        iii. **11 = register**. Ex. r1.
    d. 4-5: source operand's addressing mode, same as destination's codes.
    e. 6-9: operation code, we have 16 operations (0 – 15).
    f. 10-13: parameters addressing modes, relevant only when using jmp/bne/jsr operations. Same as source & destination codes.

4. Addressing mode

Code	Mode	Additional Words	Way of writing	Example
0	**Instant**	Contains the number in 12 bits	Operand starts with # and after that a number	mov #-1,r2
1	**Direct**	Destination of a word in memory	The operand is a label declared by .data/string/extern	x: .data 23 dec x
2	**Relative**	At most 3 words, the first is the label and the 2 others are the parameters. If both parameters are register then they will be combined into 1 word	see bits 10-13	jmp/bne/jsr L1(#r,N)
3	**Direct register**	A register, if it's a src then it will contain in bits 2-7 the register number, if it's dest than it will be bits 8-13	The operand is the name of the register	mov r1,r2

5. Operations
    a. **PC = program counter**, holds the current memory's address of the current operation (current address of the first word of the operation).
    b. Operations can handle 2 / 1 / 0 operands.
        i. When there are 1 / 0 operands the srcAdd bits are 0 or the srcAdd & the destAdd are 0 respectively.

Code	Name	Operation	Example
0	**mov**	Copy first operand(=srcOp) to second operand(=destOp)	mov A,r1
1	**cmp**	Compare 2 operands & updating Z flag in PSW to 1 if equals	cmp A,r1
2	**add**	destOp = srcOp + destOp	add A,r0
3	**sub**	destOp = destOp - destOp	sub #3,r1
4	**not**	destOp = ^destOp, reversing the bits	not r2
5	**clr**	destOp = 0, resetting the content	clr r2
6	**lea**	Load srcOp label's memory to destOp	lea HEL,r1
7	**inc**	destOp = destOp + 1	inc r2
8	**dec**	destOp = destOp - 2	dec C
9	**jmp**	PC = destOp, jump back/forward. If includes parameters, then assign them to r6 & r7.	jmp LINE(#6,r4)
10	**bne**	If (Z in PSW) = 0 then PC = destOp. Same if as jmp. Branch if not equal.	bne LINE
11	**red**	Reads stdin char's ASCII code into destOp	red r1
12	**prn**	Print destOp char's ASCII code to stdout	prn r1
13	**jsr**	push(PC) into stack and PC = destOp. Same if as jmp.	jsr FUNC
14	**rts**	PC = pop() stack, return from jsr last routine.	rts
15	**stop**	Stop the program	stop

6. Macros
    a. A code that contains a sentence.
    b. Ex.

```
mcr m1          m1          inc r2
 inc r2         ...         mov A,r1
 mov A,r1   ->  m1      ->  ...
endmcr          ...         inc r2...
```

    c. There aren't nested macros.
    d. Declaration or operation's name can't be a macro's name.
    e. Macro's definition will always be before the call for the macro and will end with an endmcr.
    f. Pre-assembler will create a file with the expansion of the macros.

7. Statements
    a. .as file composed with lines that contain statements of the language, each statement is in a different line, separated by the '\n' character.
    b. Max characters in line is 80, not including the '\n'.

Statement type	Explanation
Empty	Only spaces & tabs
Comment	The first character is ';', the program should ignore this statement.
Declarative	This statement is a directive to the assembler program. It does not generate machine instruction
Operation	This statement generates machine instruction that needs to be executed by the CPU. The statement represents machine instruction in symbolic form.

8. Declarative statement
    a. The structure is: "(optional label) (. directive name) (directive parameters)".
    b. The word that contains the declarative statement doesn't include A,R,E bits, and the coding is filled by 14 bits.

Directive	Explanation	Example
.data	Parameters are integers separated by comma. Between numbers & commas can be spaces & tabs. A comma isn't allowed before the first number & after the last number. A space is allocated in the data image & the data counter is advanced by the number of parameters. If there is a label name, the label name is assigned with the value in the data image and get inserted to the symbols table.	X: .data +7,-5, 9 mov X, r1
.string	Gets one string as a parameter, the ASCII characters are coded to their numeric ASCII values & get inserted to the data image. At the end, a 0 is inserted, to mark the end of a string. The data counter is increased by the length of the string + 1. If it gets a label the value of it will point to the location in the memory that stores the ASCII code of the first character.	STR: .string "abcd"
.entry	The parameter is the name of the label defined in the current file.	.entry W
.extern	The parameter is the name of the label defined not in the current file.	.extern A

9. Operation statement
    a. The structure is: "(optional label) (op name) (0-2 operands)".
    b. You must separate between the op name and the first operand by at least 1 space/tab.
    c. If a label is appearing at the beginning of the operation statement, then this label is inserted into the table of symbols, and its' value is pointing to the location of the operation statement in the code image.
    d. Ex.
        i. H: add r7,B
        ii. E: bne X
        iii. END: stop

10. Statements fields definitions
    a. Label
        i. A symbol at the beginning of a statement.
        ii. Starting with a lower/upper case letter and up to 29 alphanumeric letters, the max length is 30. Can't be a reserved word.
        iii. End with a ':' and must be adjacent to the label with no spaces.
        iv. The same label can't be defined more than once.
        v. Value is assigned by the context it's created, getting the **data counter** value if it's .data or .string, otherwise getting the **instruction counter**.
    b. Number – a legal integer in the base of 10 with an optional '+' / '-'.
    c. String – a sequence of ASCII characters surrounded by double quotation marks.

11. Two pass assembler
    a. In the first pass, we recognize the labels and give each label a numeric value that represents its address.
    b. In the second pass, through labels value, operation codes & registers we build the machine code.
    c. We replace the operations codes with their binary codes & the labels with their addresses.
    d. We hold a label table that holds all the labels in the file. In the first pass, we build it, in the second pass we replace the labels with their values.
    e. **Separation of data and instructions** – we organize the instructions section before the data section.
    f. **Assembler error detection**
        i. Assume that there are no errors in the macro definitions, the pre-assembler doesn't need to detect errors.
        ii. There might be errors in other places, we need to detect & print all of them to the stdout by type and line of the error.
        iii. Don't stop when an error is found, & don't make output files.

12. Format of input & output
    a. Input – Pass from cmd args, one or more. They're *.as files and will be passed as *. Ex. H.as will be H.
    b. Output
        i. **am:** src file after pre-assembler.
        ii. **ob = object:** the first line contains 2 numbers, IC & DC with one space between them. After that, each line contains the address, starting from 0100, and the machine code in special binary (='0'='.' & '1'='/') of each word.
        iii. **ext = externals:** each line contains the name and memory address of an external label in special binary. There might be serval appearance of the label in the code, each appearance should be in a different line.
        iv. **ent = entries:** each line contains the name and memory address of an internal label in special binary.
        v. If there are not .external/entry don't make the specific file.

13. Algorithms
    a. Pre-assembler
        i. Read the next line from the file, if EOF, then save the file & finish.
        ii. If the first field is macro from the macro's table, then replace it and go back to i.
        iii. If the first field is "mcr"
            1. Turn on the "isMCR" flag, save the name to the macro table & delete the line.
        iv. Read the next line from the file, if EOF, then save the file & finish.
        v. If isMCR is on
            1. If line = "endmcr"
                a. then delete the line, turn off isMCR & go back to i.
            2. Insert the line into the macro table, delete it & go back to iv.
        vi. Go back to i.
    b. First pass
        i. Initialize IC <- 0, DC <- 0.
        ii. Read line, if EOF, then update data in label table & begin 2nd pass.
        iii. If the first field is label, then turn on "isLabel" flag.
        iv. If the line is a directive of .data/string
            1. If isLabel is on, then insert the label into the label table with data type & DC value*, DC<-DC+1 and go back to ii. *If already exists, error.
        v. If the line is a directive of .entry/extern
            1. If it is a .extern, then enter each label that is an operand of this directive into the label table with extern type & without a value. Go back to ii.
        vi. If isLabel is on, then insert the label into the label table with code type & IC value*. *If already exists, error.
        vii. Search opName in opTable, If doesn't exist, error.
        viii. Analyze the operands and L<-number of words the op takes.
        ix. Build the binary code of the first word of the op.
        x. IC <- L + IC and got back to ii.
    c. Second pass
        i. Initialize IC <- 0.
        ii. Read line, if EOF, then update output files & finish.
        iii. If the first field is label, then skip it.
        iv. If the first field is .data/string/extern, then go back to ii.
        v. If the first field is .entry, then mark the corresponding labels in the label table as entry type & go back to ii.
        vi. Build each operand its binary code, if the operand is a label, then find its address in the label table.
        vii. IC <- L + IC and got back to ii.