Section 04.2 - Assembly Language

Layer 5: Assembly

Syllabus Content Section 04: Processor Fundamentals

S04.2.1 Show understanding of the relationship between assembly language and machine code

Assembly Language	Machine Code
A low-level programming language in which there is a strong correspondence between the program's statements and the architecture's machine code instructions	A Computer program written in machine language instructions that can be executed directly by a computer's central processing unit(CPU)
Follows a syntax similar to the English Language	Consists of binaries, which are zeros and ones
Understood by the programmer	Only understood by the CPU
Consists of a set of standard instruction	Depends on the platform or the operating system
Used by applications such as real-time systems, and microcontroller-based embedded systems	Can by directly executed by the CPU to perform the defined tasks in the computer program

S04.2.2 Describe the different stages of the assembly process for a two-pass assembler

Apply the two-pass assembler process to a given simple assembly language program

1. Reads all your code one time
2. Then it reads your code again and this time instead of saying “memory location 07” it actually uses the real number inside memory location 07 and puts it in a VALUE table

- Removal of comments

- Replacement of a macro name used in an instruction by the list of instruction that consistute the macro definition
- removal and storage of directives to be acted upon later.

S04.2.3 Trace a given simple assembly language program

Pep/p Assembly:

```
         BR      main
bonus:   .EQUATE 10          ;constant
exam1:   .BLOCK  2           ;global variable # 2d
exam2:   .BLOCK  2           ;global variable # 2d
score:   .BLOCK  2           ;global variable # 2d
;
main:    DECI    exam1,d     ;scanf("%d %d", &exam1, &exam2)
         DECI    exam2,d
         LDWA    exam1,d     ;score = (exam1 + exam2) / 2 + bonus
         ADDA    exam2,d
         ASRA
         ADDA    bonus,i
         STWA    score,d
         STRO    msg,d       ;printf("score = %d\n", score)
         DECO    score,d
         LDBA    '\n',i
         STBA    charOut,d
         STOP
msg:     .ASCII  "score = \x00"
         .END
```

S04.2.4 Show understanding that a set of instructions are grouped

Including the following groups:

- Data movement
- Input and output of data
- Arithmetic operations

- Unconditional and conditional instructions
- Compare instructions

Data movement

Opcode	Operand	Explanation
LDM	# n	Immediate addressing. Load the number n to ACC
LDD	“address”	Direct addressing. Load the contents of the location at the given address to ACC
LDI	“address”	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
LDV	“address”	Indexed addressing. Form the address from “address” + the contents of the index register. Copy the contents of this calculated address to ACC
LDR	# n	Immediate addressing. Load the number n to IX
MOV	“register”	Move the contents of the accumulator to the given register (IX)
STO	“address”	Store the contents of ACC at the given address

Input and output of data

Opcode	Operand	Explanation
IN		Key in a character and store its ASCII value in ACC
OUT		Output to the screen the character whose ASCII value is stored in ACC

Arithmetic operations

Opcode	Operand	Explanation
ADD	“address”	Add the contents of the given address to the ACC
ADD	# n/Bn/&n	Add the number n to the ACC
SUB	“address”	Subtract the contents of the given address from the ACC
SUB	# n/Bn/&n	Subtract the number n from the ACC
INC	“register”	Add 1 to the contents of the register (ACC or IX)
DEC	“register”	Subtract 1 from the contents of the register (ACC or IX)

Unconditional and conditional instructions

Opcode	Operand	Explanation
AND	# n/Bn/&n	Bitwise AND operation of the contents of ACC with the operand
AND	“address”	Bitwise AND operation of the contents of ACC with the contents of “address”
XOR	# n/Bn/&n	Bitwise XOR operation of the contents of ACC with the operand
XOR	“address”	Bitwise XOR operation of the contents of ACC with the contents of “address”
OR	# n/Bn/&n	Bitwise OR operation of the contents of ACC with the operand
OR	“address”	Bitwise OR operation of the contents of ACC with the contents of “address”
LSL	# n	Bits in ACC are shifted logically n places to the left. Zeros are introduced on the right hand end
LSR	# n	Bits in ACC are shifted logically n places to the right. Zeros are introduced on the left hand end

> Compare instructions

Opcode	Operand	Explanation
JMP	“address”	Jump to the given address
CMP	“address”	Compare the contents of ACC with the contents of “address”
CMP	# n	Compare the contents of ACC with number n
CMI	“address”	Indirect addressing. The address to be used is at the given address. Compare the contents of ACC with the contents of this second address
JPE	“address”	Following a compare instruction, jump to “address” if the compare was True
JPN	“address”	Following a compare instruction, jump to “address” if the compare was False

S04.2.5 Show understanding of the different modes of addressing

Including Immediate, direct, indirect, indexed, relative

Addressing mode	Use of the operand
Immediate	The operand is the value to be used in the instruction; (SUB # 48)
Direct	The operand is the address which holds the value to be used in the instruction; (ADD TOTAL)
Indirect	The operand is an address that holds the address which has the value to be used in the instruction.
Indexed	The operand is an address to which must be added the value currently in the index register(IX) to get the address which holds the value to be used in the instruction

Opcode	Operand	Explanation
LDM	# n	Immediate addressing. Load the number n to ACC
LDD	“address”	Direct addressing. Load the contents of the location at the given address to ACC
LDI	“address”	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
LDV	“address”	Indexed addressing. Form the address from “address” + the contents of the index register. Copy the contents of this calculated address to ACC
LDR	# n	Immediate addressing. Load the number n to IX
MOV	“register”	Move the contents of the accumulator to the given register (IX)
STO	“address”	Store the contents of ACC at the given address
ADD	“address”	Add the contents of the given address to the ACC
ADD	# n/Bn/&n	Add the number n to the ACC
SUB	“address”	Subtract the contents of the given address from the ACC
SUB	# n/Bn/&n	Subtract the number n from the ACC
INC	“register”	Add 1 to the contents of the register (ACC or IX)
DEC	“register”	Subtract 1 from the contents of the register (ACC or IX)
JMP	“address”	Jump to the given address
CMP	“address”	Compare the contents of ACC with the contents of “address”

Opcode	Operand	Explanation
CMP	# n	Compare the contents of ACC with number n
CMI	“address”	Indirect addressing. The address to be used is at the given address. Compare the contents of ACC with the contents of this second address
JPE	“address”	Following a compare instruction, jump to “address” if the compare was True
JPN	“address”	Following a compare instruction, jump to “address” if the compare was False
IN		Key in a character and store its ASCII value in ACC
OUT		Output to the screen the character whose ASCII value is stored in ACC
END		Return control to the operating system
AND	# n/Bn/&n	Bitwise AND operation of the contents of ACC with the operand
AND	“address”	Bitwise AND operation of the contents of ACC with the contents of “address”
XOR	# n/Bn/&n	Bitwise XOR operation of the contents of ACC with the operand
XOR	“address”	Bitwise XOR operation of the contents of ACC with the contents of “address”
OR	# n/Bn/&n	Bitwise OR operation of the contents of ACC with the operand
OR	“address”	Bitwise OR operation of the contents of ACC with the contents of “address”
LSL	# n	Bits in ACC are shifted logically n places to the left. Zeros are introduced on the right hand end
LSR	# n	Bits in ACC are shifted logically n places to the right. Zeros are introduced on the left hand end

```
<label>: <opcode> <operand> Labels an instruction
<label>:          <data>    Gives a symbolic address <label> to the memory
location
with contents <data>
```

All questions will assume there is only one general purpose register available (Accumulator)
ACC denotes Accumulator

IX denotes Index Register

<address> can be an absolute or symbolic address

# denotes a denary number, e.g. #123

B denotes a binary number, e.g. B01001010

& denotes a hexadecimal number, e.g. &4A