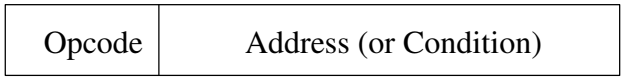
MARIE Assembly Language Supplement for Sections 4.8 - 4.10 of the textbook

Summary of the MARIE Assembly Language

Type of Instructions	Mnemonic	Hex Opcode	Description
Arithmetic	ADD X	3	Add the contents of address X to AC
	SUBT X	4	Subtract the contents of address X from the AC
	ADDI X	B	Add Indirect: Use the value at X as the actual address of the data operand to add to AC
	CLEAR	A	Put all zeros in the AC
Data Transfer	LOAD X	1	Load the contents of address X into AC
	STORE X	2	Store the contents of AC at address X
	LOADI X	D	Load Indirect: Use the value of X as the actural address of the data operand to load into the AC
	STOREI X	E	Store Indirect: Use the value of X as the actural address where the AC value is stored.
I/O	INPUT	5	Input a value from the keyboard into AC
	OUTPUT	6	Output the value in AC to the display
Branch	JUMP X	9	Unconditional branch to X by loading the value of X into PC
	SKIPCOND C	8	Skip the next instruction based on the condition, C: $C = 000_{16}$: skip if AC is negative ($b_{11}b_{10} = 00_2$) $C = 400_{16}$: skip if the AC = 0 ($b_{11}b_{10} = 01_2$) $C = 800_{16}$: skip if the AC is positive ($b_{11}b_{10} = 10_2$)
Subroutine call and return	JNS X	0	Jump-and-Store: Store the PC at address X and jump to X+1
	JUMPI X	C	Use the value **at** X as the address to jump to
	HALT	7	Terminate the program

MARIE Machine-language Instruction Format



A simple MARIE program can be written to perform the high-level language statements:

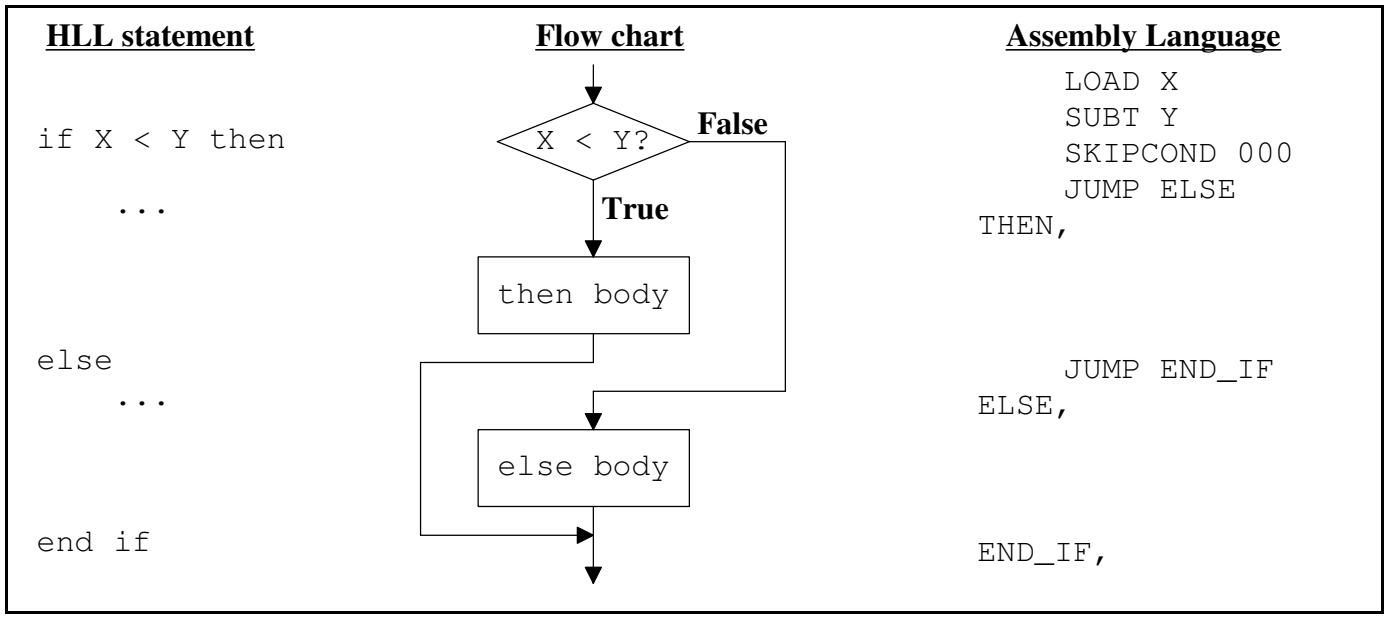
RESULT = X + Y - Z
print RESULT

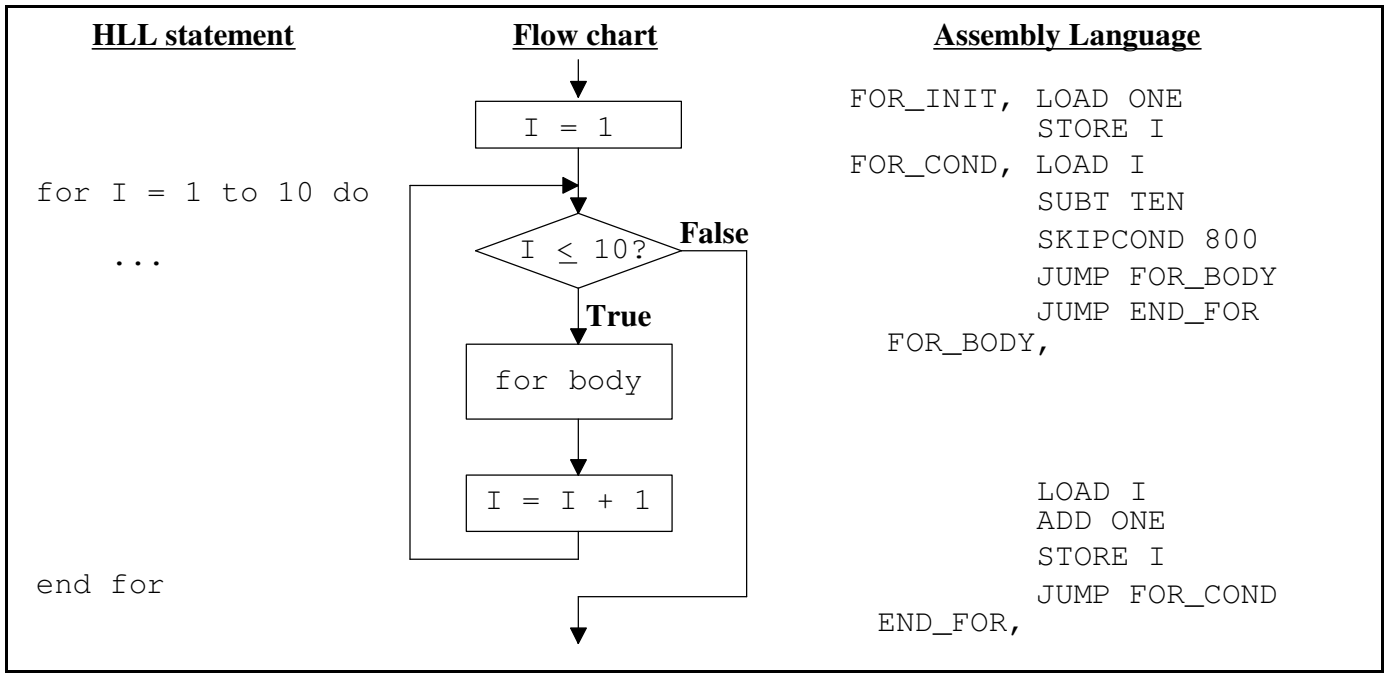
Address	Label	Assembly Language	Machine Language
0		LOAD X	1006_{16}
1		ADD Y	3007_{16}
2		SUBT Z	4008_{16}
3		STORE RESULT	2009_{16}
4		OUTPUT	6000_{16}
5		HALT	7000_{16}
6	X,	DEC 10	$000A_{16}$
7	Y,	DEC 20	0014_{16}
8	Z,	DEC 5	0005_{16}
9	RESULT,	DEC 0	0000_{16}

The lines at address 6 to 9 are *assembler directives* (directions to the assembler) to initialize the memory location associated with X (address 6) to DECimal 10, the memory location associated with Y (address 7) to 20, etc. Lines at address 0 to 5 are the actual machine-language MARIE program. If the PC = 0 (program counter), the program execution would start at address 0 which contains 1006_{16}. This instruction would be

fetched into the CPUs IR (instruction register), bits 15-12 contain the operations code of 1_{16} would be decoded to determine that it is a LOAD instruction. Execution of the LOAD causes the specified memory address's (006_{16} in bits 11-0) content to be loaded into the accumulator (AC) register (i.e., the value 10_{10} would be loaded into the AC). During the fetch-decode-execute cycle, the PC would get incremented to the next instruction. The program instructions are executed sequentially until the HALT instruction which stops the program.

The branch instructions, JUMP and SKIPCOND, potentially cause the PC to "jump" (i.e., alter the *flow of control* in the program). These instructions are useful for implementing high-level language selection (IF, IF-THEN-ELSE, SWITCH, etc.) and looping statements (FOR, WHILE, REPEAT, etc.). For example, consider the following IF-THEN-ELSE statement and corresponding flow-chart:



If X < Y is True, then the value of (X-Y) in the AC is negative. The "SKIPCOND 000" cause the JUMP ELSE instruction to be jumped over if the AC is negative. Since the then-part code follows the JUMP ELSE instruction, it is only executed if X < Y. After the then-part code is executed, the JUMP END_IF causes the else-body to be skipped. If X < Y is False, then the value of (X - Y) in the AC will not be negative the SKIPCOND 000 instruction will not jump over the JUMP ELSE instruction.

For a loop example, consider the following FOR-loop and corresponding flow-chart: