**13th February, 2019**

**PROGRAMMING WITH ASSEMBLY LANGUAGE**

**Assignment 1**

**GROUP MEMBERS**

**Jesse Anim 1644417**

**Yinbil Nayinzor Nicholas 1659717**

**Frank Sarpong 1658117**

**Paddy Gabriel 1657017**

**Manu Leonard Amankwah 1652917**

**Bamfo Emmanuel Oppong 1647417**

**Yeboah Enoch 1661117**

**Ayomide Olomi 1655117**

**Ebenezer Anane Wiredu 1659217**

**1. ABY ( -- )**

Compiles the opcode sequence for the ABY instruction into the dictionary. When later executed, this code adds the unsigned 8 bit contents of accumulator B to the contents of register Y and leaves the result in Y.

**2. ADCA ( arg\mode -- )**

Compiles the opcode sequence for the ADCA instruction into the dictionary. When later executed, this code adds the carry bit to the sum of the operand (specified by arg and mode) and the contents of accumulator A, and places the result in accumulator A.

**3. ADDB**

ADDB ( arg\mode -- )

Compiles the opcode sequence for the ADDB instruction into the dictionary. When later executed, this code adds the operand (specified by arg and mode) and the contents of accumulator B, and places the result in accumulator B.

**4. AGAIN,**

AGAIN, ( -- )

Used within a code definition to designate the end of an assembly coded infinite loop. Use as:

BEGIN, <code to be iterated>

AGAIN,

The words between BEGIN, and AGAIN, are executed indefinitely. AGAIN, is equivalent to NEVER UNTIL,

**5. ALWAYS**

ALWAYS ( -- condition )

Within a code definition, leaves a condition flag on the data stack. Indicates an "always true" condition.

**6. BCC**

BCC ( arg\mode -- | mode must be REL )

Compiles the opcode sequence for the BCC instruction into the dictionary. When later executed, this code executes a branch to the address equal to PC + arg + 2, if C in CCR is clear.

**7. BEGIN,**

BEGIN, ( -- )

Used within a code definition to designate the beginning of a looping structure. Use as:

BEGIN, . . . . condition.flag UNTIL,

or

BEGIN, . . . . condition.flag WHILE, . . . . REPEAT,

or

BEGIN, . . . . AGAIN,

The words after UNTIL, or REPEAT, are executed after the loop structure terminates. BEGIN, . . . . AGAIN, is an infinite loop.

**8. COM**

COM ( arg\mode -- )

Compiles the opcode sequence for the COM instruction into the dictionary. When later executed, this code replaces the operand specified by arg and mode with its ones complement.

**9. COMB**

COMB ( -- )

Compiles the opcode sequence for the COMB instruction into the dictionary. When later executed, this code replaces the contents of accumulator B with its ones complement (which is formed by complementing the state of each bit).

**10. BITB**

BITB ( arg\mode -- )

Compiles the opcode sequence for the BITB instruction into the dictionary. When later executed, this code modifies the bits in the condition code register according to the result of performing a logical AND between the operand (specified by arg and mode) and accumulator B. Neither the contents of accumulator B nor the operand are affected.

**11. DIR**

DIR ( -- mode )

Used within a code definition, leaves a constant on the stack indicating that the direct addressing mode should be used by an instruction opcode.

**12. ELSE,**

ELSE, ( -- )

ELSE, is used in assembly coded routines to mark the beginning of the "else portion" of a conditional structure. Use as:

condition IF, . . . .

ELSE, . . . .

ENDIF, ( or THEN,)

When executed, ELSE, causes a branch instruction to be compiled into the dictionary and resolves IF,'s branch. When the compiled code is later executed, the code between ELSE, and ENDIF, is executed if the condition is not met.

**13. ENDIF,**

ENDIF, ( -- )

ENDIF, is used in assembly coded routines to mark the end of a conditional IF, structure. ENDIF, and THEN, are synonyms. Use as:

condition IF, . . . .

ELSE, . . . .

ENDIF,

When executed, ENDIF, resolves the branch instructions used in the conditional structure. When the code compiled by the control structure is executed, the code between IF, and ELSE, is executed if the condition is true, and then control passes to the code following ENDIF,. If the condition is false, the code between ELSE, and ENDIF, is executed, and execution continues with the code following ENDIF,. An alternate form is

condition IF, . . . . ENDIF,

When the code compiled by this control structure is executed, the code between IF, and ENDIF, is executed if the condition is true, and is not executed if the condition is false

**14 . GT**

GT ( -- condition )

Within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P > Q using signed math. Alternate interpretation: Condition is true if the Z bit is clear, and the N and V bits are either both set or both clear in the condition code register.

**15. HI**

HI ( -- condition )

Within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P > Q using unsigned math. Alternate interpretation: Condition is true if C and Z in the condition code register are zero.

**16. IF,**

IF, ( condition -- )

IF, is used in assembly coded routines to mark the start of a conditional structure. Use as:

condition IF, . . . .

ELSE, . . . .

ENDIF, ( or THEN,)

or

condition IF, . . . .

ENDIF, ( or THEN,)

When the branch compiled by IF, is executed, the code following IF, is executed if the condition is met, otherwise control is transferred to the code following ELSE, or ENDIF,. (THEN, and ENDIF, are synonyms).

**17. INCB**

INCB ( -- )

Compiles the opcode sequence for the INCB instruction into the dictionary. When later executed, this code adds one to the contents of accumulator B.

**18. JSR**

JSR ( arg\mode -- )

Compiles the opcode sequence for the JSR instruction into the dictionary. When later executed, this code increments PC properly and pushes it onto the return stack. Program control is then transferred to the effective address specified by arg and mode.

**19 . LO**

LO ( -- condition )

Used inside a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q, indicates the condition P < Q using unsigned math. Alternate interpretation: Condition is true if the C (carry) bit in the condition code register is set.

**20. LSLD**

LSLD ( -- )

Compiles the opcode sequence for the LSLD instruction into the dictionary. When later executed, this code causes a logical shift left of the contents of accumulator D. The C (carry) bit in the CCR is loaded from the most significant bit of accumulator D, and 0 is shifted into the least significant bit.

**21. MI**

MI ( -- condition )

Used within a code definition, leaves a condition flag on the data stack. Used after a comparison of the form P - Q = R, indicates the condition R < 0 using signed math. Alternate interpretation: Condition is true if the N bit in the condition code register is set.

**22. MUL**

MUL ( -- )

Compiles the opcode sequence for the MUL instruction into the dictionary. When later executed, this code multiplies the contents of accumulator A by the contents of accumulator B and stores the result in accumulator D.

**23. NEVER**

NEVER ( -- condition )

Used within a code definition, leaves a condition flag on the data stack. Indicates a "never true" condition.

**24. PSHA**

PSHA ( -- )

Compiles the opcode sequence for the PSHA instruction into the dictionary. When later executed, this code stores the contents of accumulator A at the next available location on the return stack pointed to by the S register and decrements the S register. 

**25. PSHY**

PSHY ( -- )

Compiles the opcode sequence for the PSHY instruction into the dictionary. When later executed, this code stores the contents of index register Y at the next available locations on the return stack pointed to by the S register and decrements the S register by 2.

**26. REPEAT,**

REPEAT, ( -- )

REPEAT, is used to designate the end of an assembly coded structure.

BEGIN, . . . .

condition.flag WHILE, . . . .

REPEAT,

**27. ROL**

ROL ( arg\mode -- )

Compiles the opcode sequence for the ROL instruction into the dictionary. When later executed, this code shifts the bits in the operand specified by arg and mode left by one bit. The C (carry) bit is shifted into the least significant bit of arg, and the most significant bit in arg is shifted into C.

**28. RTI**

RTI ( -- )

Compiles the opcode sequence for the RTI instruction into the dictionary. When later executed, this code restores accumulators A and B, and registers X, Y and PC with values pulled from the stack.

**29. SBCB**

SBCB ( arg\mode -- )

Compiles the opcode sequence for the SBCB instruction into the dictionary. When later executed, this code subtracts the contents of the operand specified by arg and mode and the contents of C from accumulator B and stores the result in accumulator B.

**30. STAA**

STAA ( arg\mode -- )

Compiles the opcode sequence for the STAA instruction into the dictionary. When later executed, this code stores the contents of accumulator A into the effective address specified by arg and mode.

**31. TAB**

TAB ( -- )

Compiles the opcode sequence for the TAB instruction into the dictionary. When later executed, this code transfers the contents of accumulator A to accumulator B.

**32. TAP**

TAP ( -- )

Compiles the opcode sequence for the TAP instruction into the dictionary. When later executed, this code transfers the contents of accumulator A to the condition code register.

**33. THEN,**

THEN, ( -- )

THEN, is used in assembly coded routines to mark the end of a conditional IF, structure. It is a synonym for ENDIF,.

**34. UNTIL,**

UNTIL, ( condition -- )

UNTIL, designates the end of an assembly coded looping structure and resolves branch instructions according to the specified condition. Use as:

BEGIN, . . . code to be executed . . .

condition UNTIL,

If the condition is true, the loop terminates and execution continues with the code following UNTIL,. If the condition is false, looping continues and execution passes to the code following BEGIN,.

**35. VC**

VC ( -- condition )

Used within a code definition, leaves a condition flag on the data stack. The condition is true when the V (2's complement overflow) flag in the condition code register is 0.

**36. WHILE,**

WHILE, ( condition -- )

WHILE, is used in assembly coded routines to mark the beginning of a the "while true" portion of a BEGIN, . . . WHILE, . . . REPEAT, loop. Use as:

BEGIN, . . . .

condition WHILE, . . . .

REPEAT,

When executed, WHILE,causes a branch instruction to be compiled into the dictionary.

When the compiled code is executed, if the condition is true, the loop continues and the code between WHILE, and REPEAT, are executed, after which control is transferred to the code following BEGIN,. If the condition is false, the loop terminates and execution continues with the code following REPEAT,.

**37. XGDX**

XGDX ( -- )

Compiles the opcode sequence for the XGDX instruction into the dictionary. When later executed, this code exchanges the contents of accumulator D with the contents of index register X.

**38 . XGDY**

XGDY ( -- )

Compiles the opcode sequence for the XGDY instruction into the dictionary. When later executed, this code exchanges the contents of accumulator D with the contents of index register Y.

**39. VS**

VS ( -- condition )

Used within a code definition, leaves a condition flag on the data stack. The condition is true when the V (2's complement overflow) flag in the condition code register is 1.

**40 .TSTB**

TSTB ( -- )

Compiles the opcode sequence for the TSTB instruction into the dictionary. When later executed, this code subtracts zero from the contents of accumulator B and sets the condition code register's bits accordingly. Accumulator B is unaffected.

**41. STOP**

STOP ( -- )

Compiles the opcode sequence for the STOP instruction into the dictionary. When later executed, this code halts all system clocks and places the system in a minimum power consumption mode if the S bit in the CCR register is clear. If the S bit is set, STOP is disabled and acts like a NOP. Recovery is accomplished by a reset, or an active low signal on XIRQ, or a non-masked IRQ interrupt.

**42. NASM**

Stands for Netwide Assembler. A free, well documented cross platform assembler for assembly language processing.

**43. .asm**

Assembly code is saved with the .asm extention. This helps the computer identify a program as an assembly language executable.

**44. Data segment**

It is represented by .data section and the .bss. The .data section is used to declare the memory region, where data elements are stored for the program. This section cannot be expanded after the data elements are declared, and it remains static throughout the program.

The .bss section is also a static memory section that contains buffers for data to be declared later in the program. This buffer memory is zero-filled.

**45. Code segment**

It is represented by .text section. This defines an area in memory that stores the instruction codes. This is also a fixed area.

**46. Stack**

This segment contains data values passed to functions and procedures within the program.

**47. Instruction Pointer (IP)**

The 16-bit IP register stores the offset address of the next instruction to be executed. IP in association with the CS register (as CS:IP) gives the complete address of the current instruction in the code segment.

**48. Stack Pointer (SP)**

The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to be current position of data or address within the program stack.

**49. Base Pointer (BP)**

The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine. The address in SS register is combined with the offset in BP to get the location of the parameter. BP can also be combined with DI and SI as base register for special addressing.

**50. Destination Index (DI)**

It is used as destination index for string operations.