Some simple examples of assembler programs

These are presented in an order that may help incremental development of your assembler system - ensure that it can handle each program in turn before tackling the next one.

After making the assembler system with the `CMAKE` command, a command of the form

```
TEST 000
```

(note no extension) will assemble the source in 000.ASM using both the CSC201 assembler and the ASSEM assembler and compare the object code - which should be identical (although this is not an exhaustive test).

```
BEG             ; 000.ASM - empty
END


BEG             ; 001.ASM - halts
NOP
HLT
END

BEG             ; 002.ASM - simple output (one byte opcodes)
CLA             ; CPU.A = 0
OTC             ; display 0
OTI             ; display 0
INC
OTC             ; display 1
OTI             ; display 1
SHL
OTC             ; display 2
SHL
OTC             ; display 4
SHR
OTC             ; display 2
HLT
END

BEG             ; 003.ASM - hex and binary output
CLA             ; CPU.A = 0
INC
OTH             ; display 01
OTB             ; display 00000001
SHL
SHL
OTH             ; display 04
OTB             ; display 00000100
HLT
END


BEG             ; 004.ASM - simple Decimal I/O
INI             ; input a Decimal number - say 130 or 70
OTI             ; signed           -126        70
OTC             ; unsigned          130        70
OTH             ; hex               82         46
OTB             ; binary            10000010   01000110
OTA             ; character         ?          F
HLT
END


BEG             ; 005.ASM - Hex I/O
INH             ; input a Hex number, say D2 or   2F
OTI             ; signed              -46         47
OTC             ; unsigned            210         47
OTH             ; hex                 D2          1F
OTB             ; binary              11010010    00101111
OTA             ; character           0           /
HLT
END

BEG             ; 006.ASM - Binary I/O
INB             ; input a Binary number - say 10101010
OTI             ; signed       -86
OTC             ; unsigned     170
OTH             ; hex          AA
OTB             ; binary       10101010
OTA             ; character    ª
HLT
END
```

```
        BEG             ; 007.ASM - character I/O
        INA             ; input a single character - say X
        OTI             ; signed     88
        OTC             ; unsigned   88
        OTH             ; hex        58
        OTB             ; binary     01011000
        OTA             ; character  X
        HLT
        END


        BEG             ; 008.ASM - simple two-byte ops
        LDI  10         ; a = 10
        OTC             ; unsigned  10
        ADI  20         ; a = 30
        OTC             ; unsigned  30
        HLT
        END


        BEG             ; 009.ASM - simple two-byte ops
        LDI  'A'        ; a = 65 (ASCII for A)
        OTC             ; unsigned    65
        OTA             ; character   A
        LDI  0111%      ; a = 7
        OTC             ; unsigned 7
        LDI  0FFH       ; a = 255
        OTC             ; unsigned 255
        HLT
        END


        BEG             ; 010.ASM - backward reference
LOOP    LDI  '*'        ; a = 42 (ASCII for A)
        OTA             ; character  *
        BRN  LOOP       ; infinite loop!
        HLT
        END


        BEG             ; 011.ASM - a forward reference
                        ; does nothing useful - but check code carefully
LOOP    LDI  'A'        ; a = 65 (ASCII for A)
        OTA             ; character A
        BRN  STOP       ; let me out of here
        INC             ; won't get here - we have branched over this
STOP    OTC             ; unsigned  65 (not 66)
        HLT
        END


        BEG             ; 012.ASM - several forward references to one label
                        ; does nothing useful - but check code carefully
LOOP    LDI  'A'        ; a = 65
        OTA             ; character
        BRN  EXIT       ; get out of here
        INC             ; won't get here
        BRN  STOP
        DEC
        BRN  STOP
        HLT
EXIT    OTC             ; unsigned - still 65
        HLT
STOP    LDI  '?'        ; should not get here
        OTA
        BRN  LOOP       ; a valid branch, of course
        END


        BEG             ; 013.ASM - simple variable storage
        LDI  4          ; a = 4
        STA  X
        ADD  X          ; a = 8
        STA  Y
        SHL             ; a = 16
        OTC             ; unsigned 16
        LDA  Y          ; a = 8
        OTC             ; unsigned 8
        HLT
X       DS   1
Y       DS   1
        HLT
        END
```

```
        BEG             ; 014.ASM - simple constants
        LDA  AGE        ; a = 64
        OTC             ; unsigned 64
        LDA  PERIOD     ; a = 46
        OTA             ; character .
        LDA  DEVIL      ; a = 80 (ASCII for P)
        OTA             ; character P
        LDI  DEVIL      ; 14 - puzzle it out
        OTC             ; unsigned 14
        HLT
PERIOD  DC   '.'        ; American for full stop
DEVIL   DC   "PD Terry" ; you know that too
AGE     DC   64         ; you'll get there one day
IQ      DC   160        ; jealous?
        END


        BEG             ; 015.ASM - longer address fields
        LDI  100        ; puzzle it out!
        ADI  STOP
        OTC
        STA  A + 2
        LDI  5
        ADD  A + 2
        OTI
        BRN  2 + STOP
A       DS   12
STOP    LDI  A + A + A
        OTC
        HLT
        END


        BEG             ; 016.ASM - some errors
        LDI             ; puzzle them out!
        NOP
        OTC  4
        NOP
        STA  A + + 2
        NOP
        DC
        NOP
        BRN
        NOP
A       DS
        NOP
        LDI  00123%
        NOP
        LDI  123Z
        NOP
        LDI  123        Comment
        NOP
123     LDI  A + A + A
        NOP
        HLT
        END
```

The remaining examples are more substantial - the programs are designed to do something useful! There are quite a few more like this in the kit, but they are not listed out here.

```
        BEG             ; 107.ASM - write 100 asterisks to the screen
                        ; P.D. Terry, 2009

        LDI  100        ; CPU.A = 100;
        TAX             ; CPU.X = 100;
        LDI  '*'        ; CPU.A = '*';
LOOP                    ; do {
        OTA             ;   IO.writeChar('*');
        DEX             ;   CPU.X--;
        BNZ  LOOP       ; } while (CPU.X != 0);
        HLT             ; System.exit(0);
        END
```

```
        BEG             ; 109.ASM - count the bits in a number
                        ; P.D. Terry, 2009

        CLA             ;
        STA  BITS       ; bits = 0;
        INI             ; CPU.A = IO.readInt();
LOOP                    ; do {
        SHR             ;   CPU.A = CPU.A / 2
        BCC  EVEN       ;   if (CPU.A % 2 != 0) {
        STA  TEMP       ;     temp = CPU.A;      // save it
        LDA  BITS
        INC
        STA  BITS       ;     bits++;
        LDA  TEMP       ;     CPU.A = temp;      // restore it
EVEN    BNZ  LOOP       ; } while (CPU.A != 0);
        LDA  BITS       ;
        OTC             ; IO.writeCard(bits);
        HLT             ; System.exit(0);
TEMP    DS   1          ; byte temp;
BITS    DS   1          ; byte bits;
        END


        BEG             ; 114.ASM - read a word and then write it backwards
                        ; P.D. Terry, 2009

        CLX             ; CPU.X = 0;
INLOOP                  ; while (true) {
        INA             ;   CPU.A = IO.readChar();
        CPI  ' '        ;   if (CPU.A == ' ') break;
        BZE  OUTLOOP    ;
        STX  WORD       ;   word[CPU.X] = CPU.A;
        INX             ;   CPU.X++;
        BRN  INLOOP     ; }
OUTLOOP                 ; do {
        LDX  WORD-1     ;
        OTA             ;   IO.writeChar(word[CPU.X-1]);
        DEX             ;   CPU.X--;
        BNZ  OUTLOOP    ; } while (CPU.X != 0);
        HLT             ; System.exit(0);
WORD    DS   255 - *    ; // reserve memory for word
LAB
        END


        BEG             ; 118.ASM - simple program to test PRINT routine
                        ; P.D. Terry, 2009

        LDI  MESS1      ; Address of first byte of first string
        JSR  PRINT      ; Print it
        LDI  MESS2      ; Address of first byte of second string
        JSR  PRINT      ; Print that too
        HLT             ; System.exit(0);

MESS1   DC   "First exciting message$"
MESS2   DC   "Another exciting message$"
                        ;
                        ; Subroutine to print a message
                        ; To use, code JSR PRINT with CPU.A
                        ;   set up to store the address of first byte
PRINT   TAX             ; CPU.X = CPU.A to point to start of Message
LOOP                    ; while (true) {
        LDX  0          ;   CPU.A = message[CPU.X]
        CPI  '$'        ;   // String terminates with $
        BZE  CRLF       ;   if (CPU.A == '$') break;
        OTA             ;   IO.writeChar(CPU.A);
        INX             ;   CPU.X++;
        BRN  LOOP       ; }
CRLF    LDI  0DH        ;
        OTA             ; IO.writeChar(CR);
        LDI  0AH        ;
        OTA             ; IO.writeChar(LF);
        RET             ; Return when all done
        END
```

```
        BEG             ; 124.ASM - read a list of up to MAX non zero numbers and
                        ; build into a list[1..n] in ascending order
                        ; write out list forwards
                        ; P.D. Terry, 2009

        CLA
        STA  N          ; n = 0;
INLOOP                  ; while (true) {
        INI             ;   CPU.A = IO.readInt();
        BZE  SORTED     ;   if (CPU.A == 0) break;
        STA  LIST       ;   List[0] = CPU.A; // sentinel
        LDA  N
        CPI  100
        JLT  INSERT     ;   if (n >= 100) {
        LDI  STR1       ;     IO.writeString("Too many values");
        JSR  WRSTR      ;     System.exit(0);
        HLT             ;   }
INSERT  TAX             ;   x = n;
WHILE   LDX  LIST       ;   while (list[x] > list[0]) {
        CMP  LIST       ;     // CPU.A = list[x]
        BLE  STORE
        STX  LIST + 1   ;     list[x+1] = list[x];
        DEX             ;     x--;
        BRN  WHILE      ;   }
STORE   LDA  LIST
        STX  LIST + 1   ;   list[x+1] = list[0];
        LDA  N
        INC
        STA  N          ;   n++;
        BRN  INLOOP     ; }
SORTED  LDI  STR3       ; // prepare to write list in order
        JSR  WRSTR      ; IO.writeString("Sorted list is")
        JSR  WRLN       ; IO.writeLine();
        LDA  N
        BNZ  DISPLAY    ; if (n == 0) {
        LDI  STR2       ;   IO.writeString("empty");
        JSR  WRSTR      ;   System.exit(0);
        HLT             ; }
DISPLAY CLX
        INX             ; x = 1;
OUTLOOP                 ; do {
        LDX  LIST
        OTI             ;   IO.writeInt(list[x]);
        INX             ;   x++;
        LDA  N
        DEC
        STA  N          ;   n--;
        BNZ  OUTLOOP    ; } while (n != 0);
        HLT             ; System.exit(0);

WRSTR                   ; Subroutine to print a nul-terminated string
                        ; To use, code JSR WRSTR with CPU.A
                        ;   set up to store the address of first byte
        TAX             ; CPU.X = CPU.A to point to start of string
WLOOP   LDX  0          ; while (str[CPU.X] != 0) {
        BZE  WEXIT      ;
        OTA             ;   IO.writeChar(CPU.A);
        INX             ;   CPU.X++;
        BRN  WLOOP      ; }
WEXIT   RET             ; return;

WRLN    LDI  0DH        ; Subroutine to output CR/LF sequence
        OTA             ; IO.writeChar(CR);
        LDI  0AH        ;
        OTA             ; IO.writeChar(LF);
        RET             ; return;

STR1    DC   "Too many values"
        DC   0          ; nul terminated string
STR2    DC   "empty"
        DC   0          ; nul terminated string
STR3    DC   "Sorted list is"
        DC   0          ; nul terminated string
N       DS   1          ; int n;
LIST    DS   101        ; int[] list = new int[101];
        END
```

```
        BEG             ; 126.ASM - sieve algorithm for primes less than 255
                        ; P.D. Terry, 2009

ENTRY                   ;
        LDI  126
        TAX
        LDI  1
LOOP                    ; for x = 126 downto 0 {
        STX  SIEVE      ;   sieve[x] = true;
        DEX
        BPZ  LOOP       ; }
        LDI  2
        JSR  WRITE      ; 2 as a special case
        CLA
        STA  I          ; i = 0;
TEST    CPI  126        ; while (i <= 126) {
        BPZ  HALT
        TAX
        LDX  SIEVE      ;   CPU.A = sieve[I];
        BZE  INCI       ;   if (!sieve[i]) {
        LDA  I
        SHL
        ADI  3
        STA  STEP       ;     step = 2 * i + 3;
        JSR  WRITE      ;     IO.write(2 * I + 3);
        LDA  I
        STA  K          ;     k = i;
REMOVE  TAX             ;     repeat {
        CLA             ;       CPU.A = false
        STX  SIEVE      ;       sieve[k] = false
        LDA  K
        ADD  STEP
        STA  K          ;       k = k + 2 * i + 3;
        BCS  INCI       ;       // but exit if overflow
        CPI  127
        BNG  REMOVE     ;     until (k > 126); // or overflow
                        ;   }
INCI    LDA  I
        INC
        STA  I          ;   i++;
        BRN  TEST       ; }
HALT    HLT

WRITE                   ; function to write
        OTC             ; A as an unsigned number
        LDI  0DH        ; CR/LF pair for line mark
        OTA
        LDI  0AH
        OTA
        RET             ; }

I       DS   1          ; byte i, k, step
K       DS   1
STEP    DS   1
SIEVE   DS   128        ; boolean[] sieve = new boolean[128];
        END
```