```
[org 0x0100]

; start of code

mov  ax, 5              ; move the constant 5 into register ax
mov  bx, 10

add  ax, bx             ; add value of bx into the value of ax

mov  bx, 15             ; add constant 15 into the value of bx
add  ax, bx

mov  ax, 0x4c00         ; exit ..
int  0x21               ; .. is what the OS should do for me




; watch the listing carefully

; a program to add three numbers using memory variables
[org 0x0100]

    mov  ax, [num1]         ; load first number in ax
    ; mov  [num1], [num2]     ; illegal
    mov  bx, [num2]
    add  ax, bx
    mov  bx, [num3]
    add  ax, bx
    mov  [num4], ax
    mov  ax, 0x4c00
    int  0x21


num1: dw   5
num2: dw   10
num3: dw   15
num4: dw   0


; watch the listing carefully

; a program to add three numbers accessed using a single label
[org 0x0100]

    mov  ax, [num1]
    mov  bx, [num1 + 2]     ; notice how we can do arithmetic here
    add  ax, bx             ; also, why +2 and not +1?
    mov  bx, [num1 + 4]
    add  ax, bx
    mov  [num1 + 6], ax     ; store sum at num1+6
    mov  ax, 0x4c00
    int  0x21

num1:   dw  5
        dw  10
        dw  15
        dw  0
```

```
 61
 62 ; a program to add three numbers accessed using a single label
 63 [org 0x0100]
 64
 65     mov  ax, [num1]
 66     mov  bx, [num1 + 2]
 67     add  ax, bx
 68     mov  bx, [num1 + 4]
 69     add  ax, bx
 70     mov  [num1 + 6], ax
 71     mov  ax, 0x4c00
 72     int  0x21
 73
 74 num1:   dw  5,  10,  15,  0
 75
 76
 77 ; a program to add three numbers directly in memory
 78 [org 0x0100]
 79
 80     mov  ax, [num1]
 81     mov  [num1 + 6], ax    ; add this value to result
 82
 83     mov  ax, [num1 + 2]
 84     add  [num1 + 6], ax
 85
 86     mov  ax, [num1 + 4]
 87     add  [num1+6], ax
 88
 89     mov  ax, 0x4c00
 90     int  0x21
 91
 92
 93 num1:   dw  5, 10, 15, 0
 94
 95
 96 ; should have the result separate!
 97 ; let's change that!
 98
 99
100 ; a program to add three numbers using byte variables
101 [org 0x0100]
102
103     mov  ax, [num1]
104
105     mov  bx, [num1+1]
106     add  ax, bx
107
108     mov  bx, [num1+2]
109     add  ax, bx
110
111     mov  [num1+3], ax
112
113     mov  ax, 0x4c00
114     int  0x21
115
116 num1: db  5, 10, 15, 0
117
118 ; something's wrong with this code.
119 ; let's figure out what that is!
120
```

```
121
122 ; a program to add three numbers using byte variables
123 [org 0x0100]
124     ; mov  ax, 0x8787
125     ; xor  ax, ax             ; We need to make sure AX is empty! Or do we?
126
127     mov  ah, [num1]         ; Intel Sotware Developer Manual - Figure 3-5
    (Page 76)
128
129     mov  bl, [num1+1]
130     add  ah, bh
131
132     mov  bh, [num1+2]
133     add  ah, bh
134
135     mov  [num1+3], ah
136
137     mov  ax, 0x4c00
138     int  0x21
139
140 num1: db  5, 10, 15, 0
141
142
143 ; a program to add three numbers using byte variables
144 [org 0x0100]
145
146     mov  ax, 0x8787
147     xor  ax, ax          ; we need to make sure AX is empty!
148
149     mov  al, [num1]
150
151     mov  bl, [num1+1]
152     add  al, bl
153
154     mov  bl, [num1+2]
155     add  al, bl
156
157     mov  [num1+3], al
158
159
160
161     ; mov  ax, bl         ; ... assemble time error. Make sure you
    understand the error!
162
163     mov  ax, 0x4c00
164     int  0x21
165
166 num1: db  5, 10, 15, 0
167
168
169 ; a program to add three numbers using byte variables
170 [org 0x0100]
171     xor  ax, ax                ; check effect on ZF
172
173     mov  bx, num1
174
175     add  ax, [bx]
176     add  bx, 2
177
178     add  ax, [bx]
```

```
179     add  bx, 2
180
181     add  ax, [bx]
182     add  bx, 2
183
184
185     mov  [result], ax
186
187     mov  ax, 0x4c00
188     int  0x21
189
190
191     ; to turn this into an iteration, we need a couple of things:
192     ; - branching instruction
193     ; - checking constraints -- e.g. c > 0       ; Intel Sotware Developer
    Manual - Figure 3-8 (Page 80)
194
195
196 num1: dw  5, 10, 15
197 result: dw 0
198
199
200 ; a program to add three numbers using byte variables
201 [org 0x0100]
202
203      ; for (int c = 3    c > 0     c--) {
204      ;   result += data[c];
205      ;}
206
207
208
209     ; initialize stuff
210     mov  ax, 0              ; reset the accumulator
211     mov  cx, 3              ; set the iterator count
212     mov  bx, num1           ; set the base
213
214     outerloop:
215         add  ax, [bx]
216         add  bx, 2
217
218         sub  cx, 1
219         jnz  outerloop
220
221
222     mov  [result], ax
223
224     mov  ax, 0x4c00
225     int  0x21
226
227
228     ; Intel Sotware Developer Manual - EFLAGS and Instructions (Page 435)
229
230 num1: dw  5, 10, 15
231 result: dw 0
232
233
234 ; a program to add three numbers using byte variables
235 [org 0x0100]
236
237     ; initialize stuff
```

```
238     mov  ax, 0              ; reset the accumulator
239     mov  cx, 10              ; set the iterator count
240     mov  bx, 0           ; set the base
241
242     outerloop:
243         add  ax, [num1 + bx]
244         add  bx, 2
245
246         sub  cx, 1
247         jnz  outerloop
248
249
250     mov  [result], ax
251
252     mov  ax, 0x4c00
253     int  0x21
254
255
256     ; Intel Sotware Developer Manual - EFLAGS and Instructions (Page 435)
257
258 num1: dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
259 result: dw 0
260
261
262 ; a program to add three numbers using byte variables
263 [org 0x0100]
264
265     ; initialize stuff
266     mov  ax, 0              ; reset the accumulator
267     mov  bx, 0              ; set the counter
268
269     outerloop:
270         add  ax, [num1 + bx]
271         add  bx, 2
272
273         cmp bx, 20          ; sets ZF=0 when they are equal
274         jne  outerloop
275
276
277     mov  [result], ax
278
279     mov  ax, 0x4c00
280     int  0x21
281
282
283     ; Intel Sotware Developer Manual - EFLAGS and Instructions (Page 435)
284
285 num1: dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
286 result: dw 0
287
288
289 [org 0x0100]
290
291     jmp start       ; see next instructions when you haven't yet executed
    this!
292
293     num1: dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
294     result: dw 0
295
296
```

```
297
298     start:
299     ; initialize stuff
300     mov  ax, 0              ; reset the accumulator
301     mov  bx, 0              ; set the counter
302
303     outerloop:
304         add  ax, [num1 + bx]
305         add  bx, 2
306
307         cmp bx, 20          ; sets ZF=0 when they are equal
308         jne  outerloop
309
310
311     mov  [result], ax
312
313     mov  ax, 0x4c00
314     int  0x21
315
316
317
318 [org 0x0100]
319
320 jmp start
321
322 data: dw   6, 4, 5, 2
323
324
325 start:
326     mov  cx, 4                          ; make 4 passes, has to be outside
    the loop!
327
328     outerloop:
329         mov  bx, 0
330
331         innerloop:
332             mov  ax, [data + bx]
333             cmp  ax, [data + bx + 2]    ; why did we move the value to AX?
334
335             jbe  noswap                 ; if we don't have to swap, we just
    jump over the swap thing
336                                         ; think of this as the "if"
337
338                 ; the swap potion
339                 mov  dx, [data + bx + 2]
340                 mov  [data + bx + 2], ax    ; again with the AX?
341                 mov  [data + bx], dx
342
343             noswap:
344             add  bx, 2
345             cmp  bx, 6
346             jne  innerloop
347
348         ; check outer loop termination
349         sub cx, 1
350         jnz outerloop
351
352
353     ; exit system call
354     mov  ax, 0x4c00
```

```
355     int  0x21
356
357
358
359 [org 0x0100]
360
361 jmp start
362
363 data: dw   6, 2, 4, 5
364 swap: db   0    ; use this as a flag
365
366 start:
367     ; mov  cx, 4                          ; make 10 passes, has to be
    outside the loop!
368
369     outerloop:
370         mov  bx, 0
371         mov  byte [swap], 0             ; why the "byte"?
372
373         innerloop:
374             mov  ax, [data + bx]
375             cmp  ax, [data + bx + 2]    ; why did we move the value to AX?
376
377             jbe  noswap                 ; if we don't have to swap, we just
    jump over the swap thing
378
379                 ; the swap potion
380                 mov  dx, [data + bx + 2]
381                 mov  [data + bx + 2], ax    ; again with the AX?
382                 mov  [data + bx], dx
383                 mov  byte [swap], 1
384
385             noswap:
386             add  bx, 2
387             cmp  bx, 6
388             jne  innerloop
389
390         ; if we didn't swap even once, we should be done
391         cmp  byte [swap], 1     ; don't need to load this in register?
392         je   outerloop
393
394         ; check outer loop termination
395         ; sub cx, 1
396         ; jnz outerloop
397
398
399     ; exit system call
400     mov  ax, 0x4c00
401     int  0x21
402
403
404
405 [org 0x0100]
406
407 jmp start
408
409 multiplicand: db 13      ; 4-bit number, save space of 8-bits
410 multiplier:   db 5       ; 4-bit
411
412 result:       db 0       ; 8-bit result
```

```
413
414 start:
415
416     mov  cl, 4            ; how many times we need to run the loop
417     mov  bl, [multiplicand]
418     mov  dl, [multiplier]
419
420
421     checkbit:
422         shr  dl, 1        ; do the rotation so that right bit is thrown in
    CF
423         jnc  skip
424             add [result], bl     ; only add if CF IS SET
425
426
427         skip:
428         shl  bl, 1         ; always shift the multiplicand
429
430     dec  cl
431     jnz  checkbit
432
433
434     mov  ax, 0x4c00
435     int  0x21
436
437
438
439 [org 0x0100]
440
441 jmp start
442
443 num1:   dw 0x40FF    ; 4400,  40FF
444
445 dest:   dw 0x40FF
446 src:    dw 0x1001
447
448
449 start:
450
451     ; shift
452     shl byte [num1], 1
453     rcl byte [num1 + 1], 1
454
455
456
457
458
459
460
461     ; addition
462     xor ax, ax   ; clear
463
464     mov al, byte[src]
465     add byte[dest], al
466
467     mov al, [src  + 1]
468     adc byte[dest + 1], al
469
470
471
```

```
472     mov  ax, 0x4c00
473     int  0x21
474
475
476 [org 0x0100]
477
478 jmp start
479
480 multiplicand:   dw 0xC8    ; 200  =   0b 11001000
481 multiplier:     db 0x32    ; 50   =   0b 00110010
482 result:         dw 0       ; should be 10,000 = 0x2710
483
484 start:
485
486 mov  cl, 8
487 mov  dl, [multiplier]
488
489
490 checkbit:
491     shr  dl, 1
492     jnc  skip
493
494         mov  al, [multiplicand]      ; extended addition
495         add  byte [result], al
496         mov  al, [multiplicand + 1]
497         adc  byte [result + 1], al
498
499     skip:
500     shl  byte [multiplicand], 1      ; extended shift
501     rcl  byte [multiplicand + 1], 1
502
503
504     dec  cl
505     jnz  checkbit
506
507
508
509
510 ; exit syscall
511 mov  ax, 0x4c00
512 int  0x21
513
514
515
516
517
518 ; helpful bash commands
519
520 ; hex to dec
521 ; echo $((16#   F))
522
523 ; dec to hex
524 ; printf '%x\n'   15
525
526 ; bin to hex
527 ; printf '%x\n' "$((2#   110010))"
528
529 ; hex to bin
530 ; printf   '\x32'    | xxd -b | cut -d' ' -f2
531
```

```
532
533 ; Let's run a 32-bit program in Ubuntu!
534
535 ; Install NASM in Ubuntu:
536 ;   sudo apt install nasm
537
538 ; Create this code file
539
540 ; Assemble:
541 ;   nasm -f elf32 -l c05-01.lst -o c05-01.o c05-01.asm
542 ;
543 ;   We want to create a format that Linux understand
544 ; i.e. ELF format in 32-bits
545 ;   (we also create a listing file)
546 ;   Read more about ELF here: https://linux-audit.com/elf-binaries-on-linux-
    understanding-and-analysis/
547
548 ; Link with shared library that 'understands' the format: ld.so in Linux
549 ;   ld -m elf_i386 -o c05-01 c05-01.o
550
551 ; Run it:
552 ;   ./c05-01
553
554
555
556 ; Now let's discuss the code!
557
558 ; in modern OSs, programs do not start executing
559 ; "from the first instruction"
560
561 ; Instead, there is a library (ld.so) that looks for the "start symbol"
562 ; and executes from there.
563
564
565 ; a section "directive" marks the parts of a program
566 ; for the ELF format  (or whatever binary format you are using)
567 SECTION .text:
568
569 ; We mark the start for this library using the following:
570 GLOBAL _start
571
572 _start:
573   ; write the string to console
574   mov eax, 0x4        ; write syscall is 0x4
575   mov ebx, 1          ; param - std output should be used
576   mov ecx, message      ; the string to write
577   mov edx, message_length   ; the length of the string
578   int 0x80          ; invoke the system call
579
580
581   ; exit the program
582   mov eax, 0x1        ; exit system call is 0x1
583   mov ebx, 0          ; exit code is 0 (return 0)
584   int 0x80          ; Comment out and see!
585
586   ; note that int is NOT the right way to do things!
587   ; (more on this later)
588
589
590 ; data section here. We can also move it above .code
```

```
591 SECTION .data:
592   ; 0xA is new line, 0x0 is null terminator
593   message: db "Hello!",  0xA,  0x0
594   message_length: equ $-message
595 
596   ; message_length: equ 8
597   ; .... is exactly the same as
598   ; #define message_length 8
599 
600 
601 
602 ; Some useful ELF details
603 ; readelf -a c05-01.o     ; shows everything
604 
605 ; readelf -h c05-01.o   ; shows headers
606 ; readelf -S c05-01.o       ; shows sections
607 
608 ; readelf -x 2 c05-01.o   ; shows section number 2
609 ; readelf -x 2 c05-01     ; see the difference between above and this
610 
611 
612 
613 
614 
615 ; View program in GDB
616 
617 ; gdb ./c05-01
618 ; layout regs       ; shows registers and disassembled code
619 ; starti      ; start the program interactively
620 ; si        ; execute one machine instruction
621 ; quit        ; exit GDB
622 
623 
624 
625 [org 0x100]
626 
627 jmp start
628 
629 data:   dw  60, 55, 45, 50
630 swap:   db  0
631 
632 
633 bubblesort:
634     dec  cx
635     shl  cx, 1                  ; we will be jumping by 2 every time. So,
    *2
636 
637     mainloop:
638         mov  si, 0              ; use as array index
639         mov  byte[swap], 0      ; reset swap flag for this iteration
640 
641         innerloop:
642             mov  ax, [bx + si]
643             cmp  ax, [bx + si + 2]
644             jbe  noswap
645 
646                 mov  dx, [bx + si + 2]
647                 mov  [bx + si], dx
648                 mov  [bx + si + 2], ax
649                 mov  byte[swap], 1
```

```
650
651             noswap:
652             add  si, 2
653             cmp  si, cx
654             jne  innerloop
655
656         cmp  byte[swap], 1
657         je   mainloop
658
659     ret    ; notice this!!
660
661
662
663
664 start:
665     mov  bx, data
666     mov  cx, 4
667
668     ; make a function call
669     call bubblesort
670
671     ; data is now sorted!
672
673     mov ax, 0x4c00
674     int 0x21
675
676
677 [org 0x100]
678
679 jmp start
680
681 data:   dw  60, 55, 45, 50
682 swapflag:   db  0
683
684
685 swap:
686     mov  ax, [bx + si]          ; this changes ax
687     xchg ax, [bx + si + 2]
688     mov  [bx + si], ax
689
690     ret
691
692
693 bubblesort:
694     dec  cx
695     shl  cx, 1                  ; This changes cx
696
697     mainloop:
698         mov  si, 0              ; This changes si
699         mov  byte[swapflag], 0
700
701         innerloop:
702             mov  ax, [bx + si]   ; This changes ax
703             cmp  ax, [bx + si + 2]
704             jbe  noswap
705
706                call swap          ; another call here
707                mov  byte[swapflag], 1
708
709             noswap:
```

```
710             add  si, 2
711             cmp  si, cx
712             jne  innerloop
713
714         cmp  byte[swap], 1
715         je   mainloop
716
717     ret    ; notice this!!
718
719
720
721
722 start:
723     mov  bx, data
724     mov  cx, 4
725
726     ; make a function call
727     call bubblesort
728
729     ; data is now sorted!
730
731     mov ax, 0x4c00
732     int 0x21
733
734
735 [org 0x100]
736
737 jmp start
738
739 data:   dw  60, 55, 45, 50
740 swapflag:   db  0
741
742
743 swap:
744     push ax  ; -------------------------;
745     ; push cx  ; ------------------;      ;
746                                   ;      ;
747     mov  ax, [bx + si]            ;      ;
748     xchg ax, [bx + si + 2]        ;      ;
749     mov  [bx + si], ax            ;      ;
750                                   ;      ;
751     dec  cx                       ;      ;
752     ; do some storage here        ;      ;
753     ; pop cx   ; ------------------;      ;
754     pop ax   ; -------------------------;
755
756     ret
757
758
759 bubblesort:
760     push ax         ; three new pushes
761     push cx
762     push si
763
764
765     dec  cx
766     shl  cx, 1
767
768     mainloop:
769         mov  si, 0                  ; use as array index
```

```
770         mov  byte[swapflag], 0       ; reset swap flag for this iteration
771
772         innerloop:
773             mov  ax, [bx + si]
774             cmp  ax, [bx + si + 2]
775             jbe  noswap
776
777                call swap    ; another call here
778                mov  byte[swapflag], 1
779
780             noswap:
781             add  si, 2
782             cmp  si, cx
783             jne  innerloop
784
785         cmp  byte[swap], 1
786         je   mainloop
787
788
789     ; pops in reverse order
790     pop si
791     pop cx
792     pop ax
793     ret    ; notice this!!
794
795
796
797
798 start:
799     mov  bx, data
800     mov  cx, 4
801
802     ; make a function call
803     call bubblesort
804
805     ; data is now sorted!
806
807     mov ax, 0x4c00
808     int 0x21
809
810
811 [org 0x100]
812
813 jmp start
814
815 data:   dw  60, 55
816 swapflag:   db  0
817
818
819 swap:
820     push ax  ; ----------------------;
821                                      ;
822     mov  ax, [bx + si]               ;
823     xchg ax, [bx + si + 2]           ;
824     mov  [bx + si], ax               ;
825                                      ;
826     pop ax   ; ----------------------;
827
828     ret
829
```

```
830
831 bubblesort:
832     ; handle stack issue for parameters ------------
833     push bp
834     mov  bp, sp
835
836     push ax
837     push bx
838     push cx
839     push si
840
841     mov  bx, [bp + 6]   ; address of data to sort
842     mov  cx, [bp + 4]   ; number of elements to sort
843
844     ; same old code from here ----------------------
845     dec  cx
846     shl  cx, 1
847
848     mainloop:
849         mov  si, 0                    ; use as array index
850         mov  byte[swapflag], 0        ; reset swap flag for this iteration
851
852         innerloop:
853             mov  ax, [bx + si]
854             cmp  ax, [bx + si + 2]
855             jbe  noswap
856
857                call swap    ; another call here
858                mov  byte[swapflag], 1
859
860             noswap:
861             add  si, 2
862             cmp  si, cx
863             jne  innerloop
864
865         cmp  byte[swap], 1
866         je   mainloop
867
868
869     ; handle parameter stack issue at end again ------------------
870     pop si
871     pop cx
872     pop bx       ; check removal
873     ; pop ax
874     pop bp       ; bp was the first thing pushed, so last popped!
875     ; stack cleared? ---------------------------------------------
876
877     ret 4        ; what is this guy?
878
879
880
881 start:
882     mov  bx, data
883     mov  cx, 2
884
885     push bx
886     push cx
887     ; make a function call
888     call bubblesort
889
```

```
890     ; data is now sorted!
891
892     mov ax, 0x4c00
893     int 0x21
894
895
896 [org 0x100]
897
898 jmp start
899
900 data:   dw  60, 55
901 ; swapflag:   db  0            ; Globals are bad! Let's make this local.
902
903
904 swap:
905     push ax  ; ----------------------;
906                                     ;
907     mov  ax, [bx + si]              ;
908     xchg ax, [bx + si + 2]          ;
909     mov  [bx + si], ax              ;
910                                     ;
911     pop ax   ; ----------------------;
912
913     ret
914
915
916 bubblesort:
917     ; handle stack issue for parameters ------------
918     push bp
919     mov  bp, sp
920
921     sub sp, 2           ; make space on the stack, just below BP
922                         ; only if you want to do local variables
923
924
925     push ax
926     push bx
927     push cx
928     push si
929
930     mov  bx, [bp + 6]   ; address of data to sort
931     mov  cx, [bp + 4]   ; number of elements to sort
932
933     ; same old code from here ----------------------
934     dec  cx
935     shl  cx, 1
936
937     mainloop:
938         mov  si, 0                      ; use as array index
939         ; mov  byte[swapflag], 0       ; reset swap flag for this iteration
940         mov  word [bp - 2], 0             ; has to be a word
941
942         innerloop:
943             mov  ax, [bx + si]
944             cmp  ax, [bx + si + 2]
945             jbe  noswap
946
947                call swap    ; another call here
948                ; mov  byte[swapflag], 1
949                mov  word [bp - 2], 1
```

```
950
951             noswap:
952             add  si, 2
953             cmp  si, cx
954             jne  innerloop
955
956         cmp  word [bp - 2],  1
957         je   mainloop
958
959
960     ; handle parameter stack issue at end again ------------------
961     pop si
962     pop cx
963     pop bx
964     pop ax
965
966     mov sp, bp   ; sp should be restored
967
968     pop bp       ; bp was the first thing pushed, so last popped!
969     ; stack cleared? ----------------------------------------------
970
971     ret 4        ; what is this guy?
972
973
974
975 start:
976     mov  bx, data
977     mov  cx, 2
978
979     push bx
980     push cx
981     ; make a function call
982     call bubblesort
983
984     ; data is now sorted!
985
986     mov ax, 0x4c00
987     int 0x21
988
989
990 [org 0x0100]
991
992 mov  ax, 0xb800         ; video memory base
993 mov  es, ax             ; cannot move to es through IMM
994 mov  di, 0              ; top left location
995
996 nextpos:
997     mov  word [es:di], 0x0776       ; 0x07 -- full white  (try 41)
998                                     ; 0x20 is the space character
999     add  di, 2
1000    cmp  di, 4000
1001    jne  nextpos
1002
1003    mov  ax, 0x4c00
1004    int  0x21
1005
1006
1007 [org 0x0100]
1008
1009    jmp start
```

```
1010
1011 message:     db   'hello world'
1012 length:      dw   11
1013
1014 clrscr:
1015     push es
1016     push ax
1017     push di
1018
1019     mov  ax, 0xb800
1020     mov  es, ax
1021     mov  di, 0
1022
1023     nextloc:
1024         mov  word [es:di], 0x0720
1025         add  di, 2
1026         cmp  di, 4000
1027         jne  nextloc
1028
1029     pop  di
1030     pop  ax
1031     pop  es
1032     ret
1033
1034
1035 printstr:
1036     push bp
1037     mov  bp, sp
1038     push es
1039     push ax
1040     push cx
1041     push si
1042     push di
1043
1044     mov ax, 0xb800
1045     mov es, ax
1046     mov di, 0
1047
1048
1049     mov si, [bp + 6]
1050     mov cx, [bp + 4]
1051     mov ah, 0x07 ; only need to do this once
1052
1053     nextchar:
1054         mov al, [si]
1055         mov [es:di], ax
1056         add di, 2
1057         add si, 1
1058
1059         ; dec cx
1060         ; jnz nextchar
1061
1062         ; alternatively
1063         loop nextchar
1064
1065
1066     pop di
1067     pop si
1068     pop cx
1069     pop ax
```

```
1070     pop es
1071     pop bp
1072     ret 4
1073
1074
1075 start:
1076     call clrscr
1077
1078     mov ax, message
1079     push ax
1080     push word [length]
1081     call printstr
1082
1083
1084
1085     ; wait for keypress
1086     mov ah, 0x1        ; input char is 0x1 in ah
1087     int 0x21
1088
1089     mov ax, 0x4c00
1090     int 0x21
1091
1092
1093 [org 0x0100]
1094
1095     jmp start
1096
1097 clrscr:
1098     push es
1099     push ax
1100     push di
1101
1102     mov  ax, 0xb800
1103     mov  es, ax
1104     mov  di, 0
1105
1106     nextloc:
1107         mov  word [es:di], 0x0720
1108         add  di, 2
1109         cmp  di, 4000
1110         jne  nextloc
1111
1112     pop  di
1113     pop  ax
1114     pop  es
1115     ret
1116
1117
1118 printnum:
1119     push bp
1120     mov  bp, sp
1121     push es
1122     push ax
1123     push bx
1124     push cx
1125     push dx
1126     push di
1127
1128     ; first, let's split digits and push them onto the stack
1129
```

```
1130     mov ax, [bp+4]   ; number to print
1131     mov bx, 10       ; division base 10
1132     mov cx, 0        ; total digit counter
1133
1134     nextdigit:
1135         mov dx, 0    ; zero out
1136         div bx       ; divides ax/bx .. quotient in ax, remainder in dl
1137         add dl, 0x30 ; convert to ASCII
1138         push dx      ; push to stack for later printing
1139         inc cx       ; have another digit
1140         cmp ax, 0    ; is there something in quotient?
1141         jnz nextdigit
1142
1143     ; now let's do the printing
1144
1145     mov ax, 0xb800
1146     mov es, ax
1147
1148     mov di, 0
1149     nextpos:
1150         pop dx          ; digit to output. Already in ASCII
1151         mov dh, 0x04    ; why is this inside the loop here?
1152         mov [es:di], dx
1153         add di, 2
1154         loop nextpos    ; cx has already been set, use that
1155         ;dec cx
1156         ;jnz nextpos
1157
1158     pop di
1159     pop dx
1160     pop cx
1161     pop bx
1162     pop ax
1163     pop es
1164     pop bp
1165     ret 2
1166
1167
1168
1169 start:
1170     call clrscr
1171
1172     mov ax, 452
1173     push ax
1174     call printnum
1175
1176
1177     ; wait for keypress
1178     mov  ah, 0x1        ; input char is 0x1 in ah
1179     int 0x21
1180
1181     mov ax, 0x4c00
1182     int 0x21
1183
1184
1185 [org 0x0100]
1186     jmp  start
1187
1188 clrscr:
1189     push es
```

```
1190     push ax
1191     push cx
1192     push di
1193
1194     mov  ax, 0xb800            ; same as before
1195     mov  es, ax
1196
1197     xor  di, di                ; starting at index 0
1198
1199     mov  ax, 0x0720            ; what to write
1200     mov  cx, 2000              ; how many times to write
1201                                ; holds the count, NOT bytes!
1202
1203     cld                        ; auto-increment
1204     rep stosw                  ; automatically writes starting from [es:di]
1205
1206     pop di
1207     pop  cx
1208     pop  ax
1209     pop  es
1210     ret
1211
1212 start:
1213     call clrscr
1214     mov  ax, 0x4c00
1215     int  0x21
1216
1217
1218
1219 [org 0x0100]
1220
1221 jmp  start
1222
1223 clrscr:
1224     push es
1225     push ax
1226     push cx
1227     push di
1228
1229     mov  ax, 0xb800
1230     mov  es, ax
1231     xor  di, di
1232     mov  ax, 0x0765
1233     mov  cx, 2000
1234
1235     cld                ; auto-increment mode
1236     rep stosw          ; rep cx times, store words
1237                        ; source is ax for word, al for bytes
1238                        ; destination is es:di
1239                        ; inc/dec di as well by 2 bytes
1240
1241     pop di
1242     pop  cx
1243     pop  ax
1244     pop  es
1245     ret
1246
1247
1248 start:
1249
```

```
1250     call clrscr
1251     mov  ax, 0x4c00
1252     int  0x21
1253
1254
1255 [org 0x0100]
1256
1257     jmp start
1258
1259 message: db 'hello world', 0
1260
1261 clrscr:
1262     push es
1263     push ax
1264     push cx
1265     push di
1266
1267     mov  ax, 0xb800
1268     mov  es, ax
1269     xor  di, di
1270     mov  ax, 0x0720
1271     mov  cx, 2000
1272
1273     cld                 ; auto-increment mode
1274     rep stosw           ; rep cx times, store words
1275                         ; source is ax for word, al for bytes
1276                         ; destination is es:di
1277                         ; inc/dec di as well by 2 bytes
1278
1279     pop  di
1280     pop  cx
1281     pop  ax
1282     pop  es
1283     ret
1284
1285 printnum:
1286     push bp
1287     mov  bp, sp
1288     push es
1289     push ax
1290     push bx
1291     push cx
1292     push dx
1293     push di
1294
1295     ; first, let's split digits and push them onto the stack
1296
1297     mov ax, [bp+4]   ; number to print
1298     mov bx, 10       ; division base 10
1299     mov cx, 0        ; total digit counter
1300
1301     nextdigit:
1302         mov dx, 0    ; zero out
1303         div bx       ; divides ax/bx .. quotient in ax, remainder in dl
1304         add dl, 0x30 ; convert to ASCII
1305         push dx      ; push to stack for later printing
1306         inc cx       ; have another digit
1307         cmp ax, 0    ; is there something in quotient?
1308         jnz nextdigit
1309
```

```
1310     ; now let's do the printing
1311
1312     mov ax, 0xb800
1313     mov es, ax
1314
1315     mov di, 0
1316     nextpos:
1317         pop dx          ; digit to output. Already in ASCII
1318         mov dh, 0x07    ; why is this inside the loop here?
1319         mov [es:di], dx
1320         add di, 2
1321         loop nextpos    ; cx has already been set, use that
1322
1323     pop di
1324     pop dx
1325     pop cx
1326     pop bx
1327     pop ax
1328     pop es
1329     pop bp
1330     ret 2
1331
1332
1333 strlen:
1334     push bp
1335     mov  bp,sp
1336     push es
1337     push cx
1338     push di
1339
1340     les  di, [bp+4]     ; load DI from BP+4 and ES from BP+6
1341     mov  cx, 0xffff     ; maximum possible length
1342
1343     xor  al, al         ; value to find
1344     repne scasb         ; repeat until scan does not become NE to AL
1345                         ; decrement CX each time
1346
1347     mov  ax, 0xffff
1348     sub  ax, cx         ; find how many times CX was decremented
1349
1350     dec  ax             ; exclude null from the length
1351
1352     pop  di
1353     pop  cx
1354     pop  es
1355     pop  bp
1356     ret  4
1357
1358
1359 start:
1360     call clrscr
1361
1362     push ds
1363     mov  ax, message
1364     push ax
1365     call strlen          ; return value is in AX
1366
1367     push ax
1368     call printnum        ; print out the length
1369
```

```
1370
1371     mov  ah, 0x1
1372     int 0x21
1373     mov  ax, 0x4c00
1374     int 0x21
1375
1376
1377 SECTION .DATA
1378   hello:     db 'Hello from ASM!',10
1379   helloLen:  equ $-hello
1380
1381 SECTION .TEXT
1382 GLOBAL say_hi
1383
1384
1385 say_hi:
1386     mov rax, rdi          ; first param goes in RDI
1387   push rax          ; save the value sent to us
1388
1389   mov eax, 4            ; write()
1390   mov ebx, 1            ; STDOUT
1391   mov ecx, hello
1392   mov edx, helloLen
1393
1394   int 80h              ; Interrupt
1395
1396   pop rax        ; get the value sent to us
1397   inc rax        ; increment it
1398   ret                  ; return val is in rax
1399
1400
1401
1402 # Assemble using: nasm -f elf64 c09-01.asm  -o c09-01-asm.o
```