hello_world.asm

```
section .data
      output: db 'Hello World', 0xa, 0xd  ; declare output with endline
(0xa, 0xd)
      outputLen: equ $-output       ; declare the length of the output

section .text
      global _start

_start:
      mov eax, 1              ; eax = 1 = sys_write
      mov edi, 1              ; edi = 1 = stdout
      mov rsi, output         ; rsi = output = label you want to print
      mov edx, outputLen      ; edx = outputLen = length of output
      syscall

      mov eax, 60       ; eax = 60 = sys_exit (exit the program correctly)
      mov edi, 0        ; edi = 0 = no error (error code)
      syscall
```

Program Template

```
section .data
      ; put variables where you know the value here
      ; you can also put variables where you don't know the value
      ; yet here if you give it a random value you'll change later

section .bss
      ; put variables where you don't know the value yet here
      ; this section is optional if you gave your unknown variables a
random value to change later

section .text
      global _start ; this tells your program where to actually start. you
need to define _start

_start:
      ; put your actual code here
      ; some examples include getting user input (like for a hangman
letter), printing data,
      ; loops (everything except declaring variables from your goto c)
```

Common Mistakes when Writing Assembly

- Using a variable instead of a register
  - All the commands except for mov can only be used with registers. If you need a value from a variable, move it into a register first. Then, move it back. Ex:

```
section .data
	myVariable: db 5

section .text
	global _start

_start:
	; CORRECT
	mov   ebx, myVariable   ; ebx = myVariable = 5
	add   ebx, 7            ; ebx = ebx + 7 = 12
	mov   myVariable, ebx   ; myVariable = ebx = 12

	; INCORRECT AND WON'T RUN
	add   myVariable, 7          ; ERROR
```

- 'Hardcoding' your addresses
  - All addresses are random when you start your program. Therefore, you can never know what they will be when you're writing it. To access the address of a variable, you always have to use the name you gave it.

```
section .data
	myVariable: db 'this is my variable'

section .text
	global _start

_start:
	; CORRECT
	mov   ebx, myVariable   ; ebx is the address of myVariable
	mov   ecx, [myVariable] ; ecx is myVariable[0] or the
character 't'

	; INCORRECT
	mov   ebx, 0x3239fa03   ; this looks like an address but
it's just something I made up
	mov   ecx, [0x3239fa03] ; this will either throw an error
or just move garbage into ecx
```

- Moving a larger register into a smaller register
    - If you look back at your diagram, you'll notice that the registers like ax, bx etc are smaller than eax and ebx. You'll get an error if you try to move a big register into a smaller one, like this:

```
_start:
      mov ebx, 5
      mov bx, ebx       ; ERROR, ebx doesn't fit in bx!
```

- Using too small a register for array access
    - NASM syntax doesn't allow you to use anything smaller than a 32 bit register for array access. That means you can only use registers that start with an e or an r

```
section .data
      myVariable: db 'hangman'

section .text
      global _start

_start:
      ; CORRECT
      mov ebx, 3
      mov [myVariable + ebx], 'y'   ; change hangman to hanyman

      ; INCORRECT - register is too small
      mov bx, 3
      mov [myVariable + bx], 'y'    ; will throw an error
because bx is too small
```