



Slide	Presenters	Topic
-------	------------	-------

# ;Assembly Language

Marieke Thomas, Ben Eckley, Dave Ciolino-Volano  
Ed Hawkins, Taylor Grant-Knight, Amanda Lee



COVER.HTML



## ;Agenda



1. Intro
2. History of Assembly
3. Fundamentals of Assembly
4. Fun with Assembly! (Code Along)
5. More Fun with Assembly!!! (HW)
6. Additional Resources
7. Questions???
8. And now a word from next week's group...



AGENDA.HTML

Ed

# ;Assembly: The code that tells your hardware what to do

Abstraction begets abstraction begets abstraction...turtles all the way down.

**Mark 1:** 5 tons, 816 cubic feet 1600 vacuum tubes, 5 horsepower (?!?!?)

**TRADIC:** First transistorized computer, 800 transistors, only .13 horseys needed

**Modern CPU:** ~290 million transistors and same # of horseys as TRADIC

## Evolution of Computers



**Abacus**  
(3000 BC)



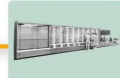
**Napier Bones**  
(1617)



**Pascaline**  
(1642)



**Leibniz Wheel**  
(1685)



**Mark I**  
(1944)



**Census Machine**  
(1889)

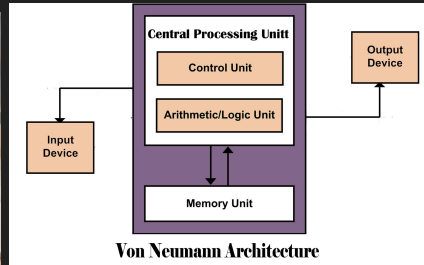
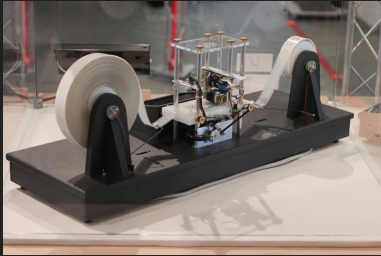


**Analytical Engine**  
(1833)

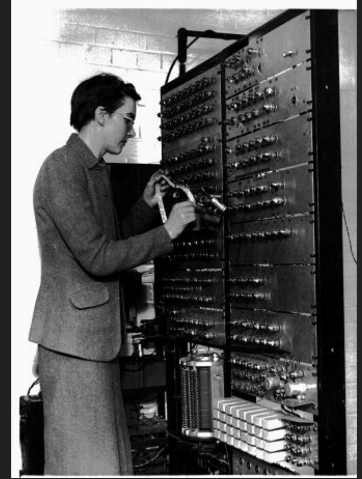


**Jacquard Loom**  
(1804)

# An incredibly brief history of Assembly part 1



Turing to Von Neumann to Kathleen Booth...but boy oh boy it goes a lot further back than that.



## An incredibly brief history of Assembly part 2

Kathleen Booth's Contracted Notation

- |     |                              |   |
|-----|------------------------------|---|
| 11) | $M \times R \rightarrow cA.$ | Clear accumulator, multiply M by R and place L.H. 39 digits of answer in A and R.H. 39 digits in R. |
| 12) | $A \div M \rightarrow cR.$   | Clear register, divide A by M, leave quotient in R and remainder in A.                              |
| 13) | $C \rightarrow M_1.$         |   |
| 14) | $C \rightarrow M_T.$         |   |
| 15) | $Cc \rightarrow M_1.$        | If number in A $\geq 0$ shift control to $M_1.$   |
| 16) | $Cc \rightarrow M_T.$        |   |
| 17) | $A \rightarrow M.$           |   |



# ;NASM Assembly

Used with Intel x86 architecture (most desktops/laptops, NOT smartphones/tablets/Apple M1 or M2 chips)

Easily connects to Linux OS

[Online compiler](#)



INDEX.HTML

Ben

Assembly is hardware-specific. There are many different assembly languages. Today we will be programming in NASM assembly

Online compiler: [https://www.tutorialspoint.com/compile\\_assembly\\_online.php](https://www.tutorialspoint.com/compile_assembly_online.php)



# ;Hello World



COMMENTS  
START WITH ;

```
1 section .text
2     global _start           ;must be declared for using gcc
3     _start:                 ;tell linker entry point
4     mov edx, len            ;message length
5     mov ecx, msg            ;message to write
6     mov ebx, 1              ;file descriptor (stdout)
7     mov eax, 4              ;system call number (sys_write)
8     int 0x80                ;call kernel
9     mov eax, 1              ;system call number (sys_exit)
10    int 0x80                 ;call kernel
11
12 section .data
13
14 msg db 'Hello, world!',0xa ;our dear string
15 len equ $ - msg            ;length of our dear string
16
```

INDEX.HTML

Taylor/Amanda

Link: [https://www.tutorialspoint.com/compile\\_assembly\\_online.php](https://www.tutorialspoint.com/compile_assembly_online.php)

Presentation: Give link to the compiler. Introduce sections (.text, .data, .bss not shown), explain that all this is for Hello World. Marieke will walk through what "mov" command is. After that, we will walk through each line of code.



# ;Assembly Sections

```
mov eax, 1
```

## section .text

Contains the actual code  
(editor)

```
msg db "a"
```

## section .data

Declares **initialized**  
variables

```
num resb 1
```

## section .bss

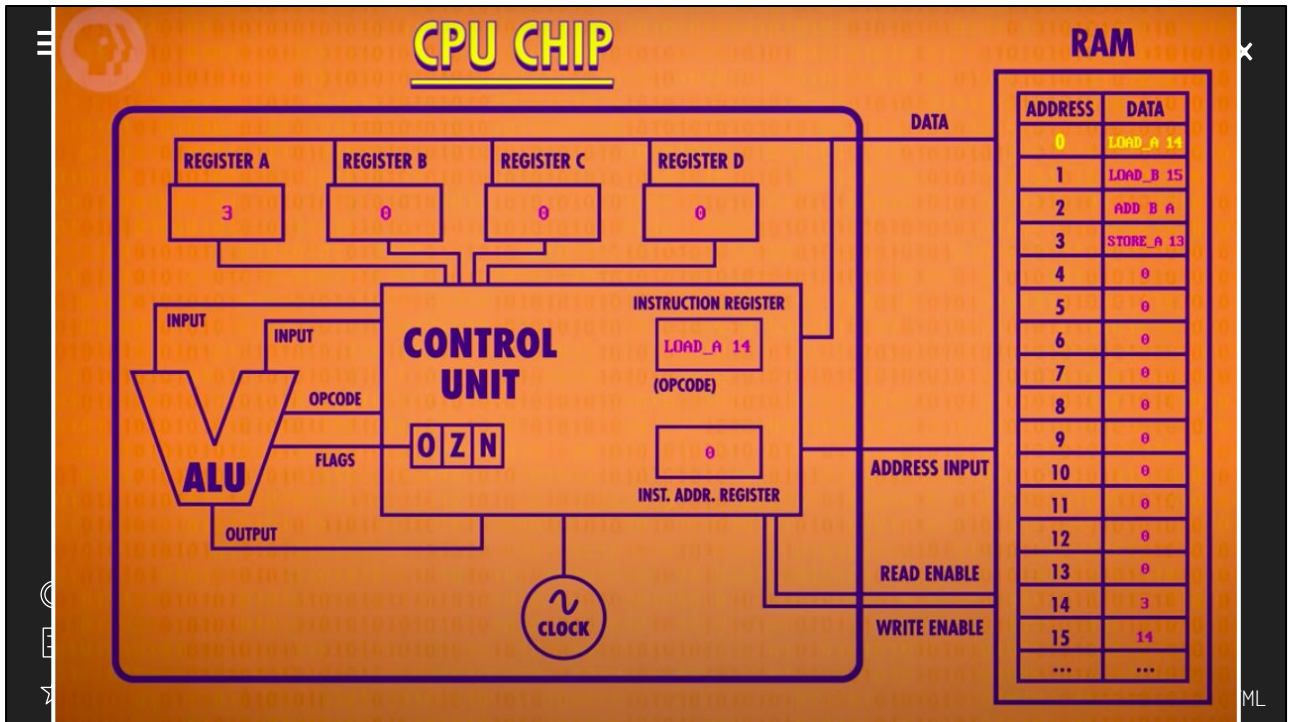
Declares **uninitialized**  
variables



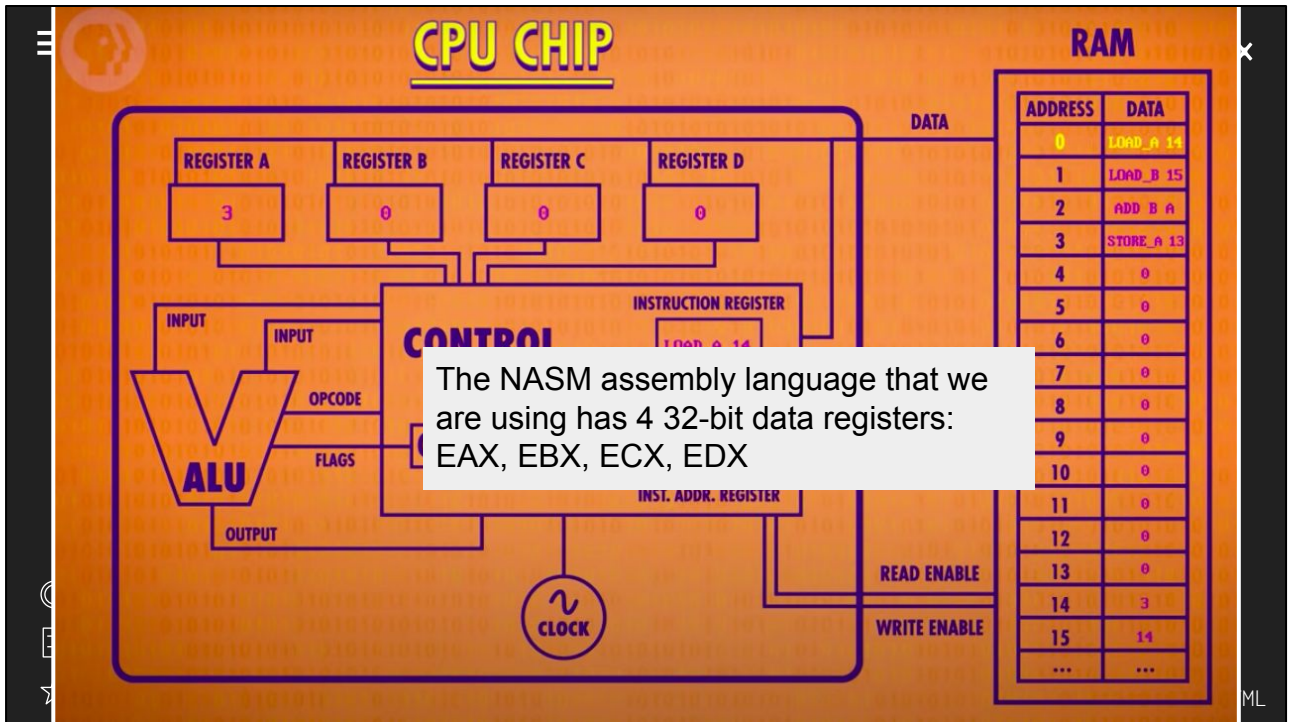
INDEX.HTML

Taylor/Amanda





Marieke RAM stores data (BOTH the instructions/code, and the info we want to remember, like strings). Inside the CPU is a very small amount of memory in the form of registers. The data stored in the registers can be accessed much more quickly than the data in RAM, so it's common for a program to load data from RAM into the registers, do something with it, and then store the result back in RAM. We do this using the mov operation



Marieke RAM stores data (BOTH the instructions/code, and the info we want to remember, like strings). Inside the CPU is a very small amount of memory in the form of registers. The data stored in the registers can be accessed much more quickly than the data in RAM, so it's common for a program to load data from RAM into the registers, do something with it, and then store the result back in RAM. We do this using the mov operation



## ;Moving data

`mov destination, source`

Moves data from the source to the destination

Example:

`mov eax, 4`

Stores the number 4 in register A

`mov eax, ebx`

Copies the data from register B into register A



Marieke



## ;Assembly Variables

Variables are declared in the .data or the .bss section

To declare a variable:

Variable-name	declaration	value
Ex:		
message	db	"hello world"
counter	db	0



INDEX.HTML

Marieke

Db = define byte

I found this thread helpful for figuring out the differences:

<https://stackoverflow.com/questions/48608423/what-do-square-brackets-mean-in-x86-assembly>



## ;Assembly Variables and Addresses



variable	Refers to the memory location of the variable
[variable]	Refers to the data stored at the memory location of the variable
register	Refers to the actual register storage space
[register]	Refers to the data stored at the memory location in the register



INDEX.HTML

Marieke

A variable doesn't actually store your data. It stores the location in RAM where the data is located.

I found this thread helpful for figuring out the differences:

<https://stackoverflow.com/questions/48608423/what-do-square-brackets-mean-in-x86-assembly>

Code: mov eax 4

Initial

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Final

Register A

4

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Marieke

Code: mov [eax] 4

Initial

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Final

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	4
4	114
5	89

Marieke

Code: mov eax ebx

Predict!

Initial

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Final

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89



Code: mov eax ebx

Initial

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Final

Register A

1

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Code: mov eax [ebx]

Predict!

Initial

Final

Register A

3

Register B

1

RAM	
Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Register A

3

Register B

1

RAM	
Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Code: mov eax [ebx]

Initial

Register A

3

Register B

1

RAM	
Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Final

Register A

42

Register B

1

RAM	
Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Code: mov [eax] ebx

Predict!

Initial

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Final

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Code: mov [eax] ebx

Initial

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Final

Register A

3

Register B

1

RAM

Memory Address	Data
1	42
2	13
3	1
4	114
5	89

Code: mov [eax] [ebx]

Predict!

Initial

Final

Register A

3

Register B

1

RAM	
Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Register A

3

Register B

1

RAM	
Memory Address	Data
1	42
2	13
3	2
4	114
5	89

Code: mov [eax] [ebx]

Initial

Register A  
3

Register B  
1

RAM	
Memory Address	Data
1	42
2	13
3	42
4	114
5	89

Final

Register A  
3

Register B  
1

RAM	
Memory Address	Data
1	42
2	13
3	42
4	114
5	89

Error! Cannot move from memory to memory!

Code: mov eax x

Predict!

Initial

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Final

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89



Code: mov eax x

Initial

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Final

Register A

2

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Code: mov eax [x]

Predict!

Initial

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Final

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Code: mov eax [x]

Initial

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Final

Register A

13

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Code: mov [eax] [x]

Predict!

Initial

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Final

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Code: mov [eax] [x]

Predict!

Initial

Final

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
x	1	42
y	2	13
	3	2
	4	114
	5	89

Error! Cannot move from memory to memory!

Register B

1

RAM

Variable	Memory Address	Data
x	1	42
y	2	13
	3	2
	4	114
	5	<del>89</del>

How can we put a 13 into memory address 5?

Instead of: ~~mov [eax] [x]~~  
mov ebx [x]  
mov [eax] ebx

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	89

Register A

5

Register B

13

RAM

Variable	Memory Address	Data
	1	42
x	2	13
y	3	2
	4	114
	5	13

☰

☐

×

☉

☐

☆

# ;Hello World

```
1 section .text
2     global _start           ;must be declared for using gcc
3     _start:                 ;tell linker entry point
4     mov edx, len            ;message length
5     mov ecx, msg            ;message to write
6     mov ebx, 1              ;file descriptor (stdout)
7     mov eax, 4              ;system call number (sys_write)
8     int 0x80                ;call kernel
9     mov eax, 1              ;system call number (sys_exit)
10    int 0x80                ;call kernel
11
12 section .data
13
14 msg db 'Hello, world!',0xa ;our dear string
15 len equ $ - msg            ;length of our dear string
16
```

INDEX.HTML

Taylor/Amanda

Printing (lines 4-8) is essentially an API call with 4 parameters, that need to be stored in the 4 registers (EAX, EBX, ECX, and EDX).

db = define byte

0xa = hex new line character

\$ = the current location

0 stdin, 1 stdout and 2 stderr

<https://stackoverflow.com/questions/17074663/need-guidance-on-understanding-basi-c-assembly>



## ;Printing in NASM Assembly

Printing is an API call to the kernel (the Linux operating system).  
The 4 data registers must be set to:

EAX	4
EBX	1
ECX	Memory address of the message to be printed
EDX	Length (# characters) of the message to be printed

Once these values are set, call the API using:  
`int 0x80`



INDEX.HTML

Taylor/Amanda





## ;Code-Along: Adding 2 Numbers

```
1 section .text
2     global _start      ;must be declared for using gcc
3     _start:            ;tell linker entry point
4
5     ;move the data to registers
6     mov eax, [num1]
7     mov ebx, [num2]
8
9     ;add the data and store to register A (eax)
10    add eax, ebx
11    add eax, '0' ;converts from number to ASCII value of
        number
12    mov [result], eax
13
14    ;print message
15    mov edx, len      ;message length
16    mov ecx, msg       ;message to write
17    mov ebx, 1         ;file descriptor (stdout)
18    mov eax, 4         ;system call number (sys_write)
19    int 0x80           ;call kernel
20
21    ;print sum
22    mov edx, 1         ;message length
23    mov ecx, result    ;message to write
24    mov ebx, 1         ;file descriptor (stdout)
25    mov eax, 4         ;system call number (sys_write)
26    int 0x80           ;call kernel
27
28
29    ;End connection to linux system
30    mov eax, 1         ;system call number (sys_exit)
31    int 0x80           ;call kernel
32
33 section .data
34     msg db "The sum is "
35     len equ $ - msg
36
37     num1 db 3
38     num2 db 4
39
40
41 section .bss                ;Uninitialized data
42     result resb 1
```

Marieke

Db = define byte (for constant)

Resb = reserve byte (for variable)

section.text

```
    global _start      ;must be declared for using gcc
_start:                ;tell linker entry point
```

;move the data to registers

```
mov eax, [num1]
```

```
mov ebx, [num2]
```

;add the data and store to register A (eax)

```
add eax, ebx
```

```
add eax, '0'
```

```
mov [result], eax
```

;print message

```
mov     edx, len      ;message length
```

```
mov     ecx, msg       ;message to write
```

```
mov     ebx, 1         ;file descriptor (stdout)
```

```
mov     eax, 4         ;system call number (sys_write)
```

```
int     0x80           ;call kernel
```

```
;print sum
    mov     edx, 1    ;message length
    mov     ecx, result ;message to write
    mov     ebx, 1    ;file descriptor (stdout)
    mov     eax, 4    ;system call number (sys_write)
    int     0x80      ;call kernel
```

```
;End connection to Linux system
    mov     eax, 1    ;system call number (sys_exit)
    int     0x80      ;call kernel
```

```
section .data
    msg db "The sum is "
    len equ $ - msg
```

```
num1 db 3
num2 db 4
```

```
section .bss      ;Uninitialized data
    result resb 1
```



;What's going on here?



INDEX.HTML

Taylor/Amanda



;What happens if you try to add 6 and 8? Why?

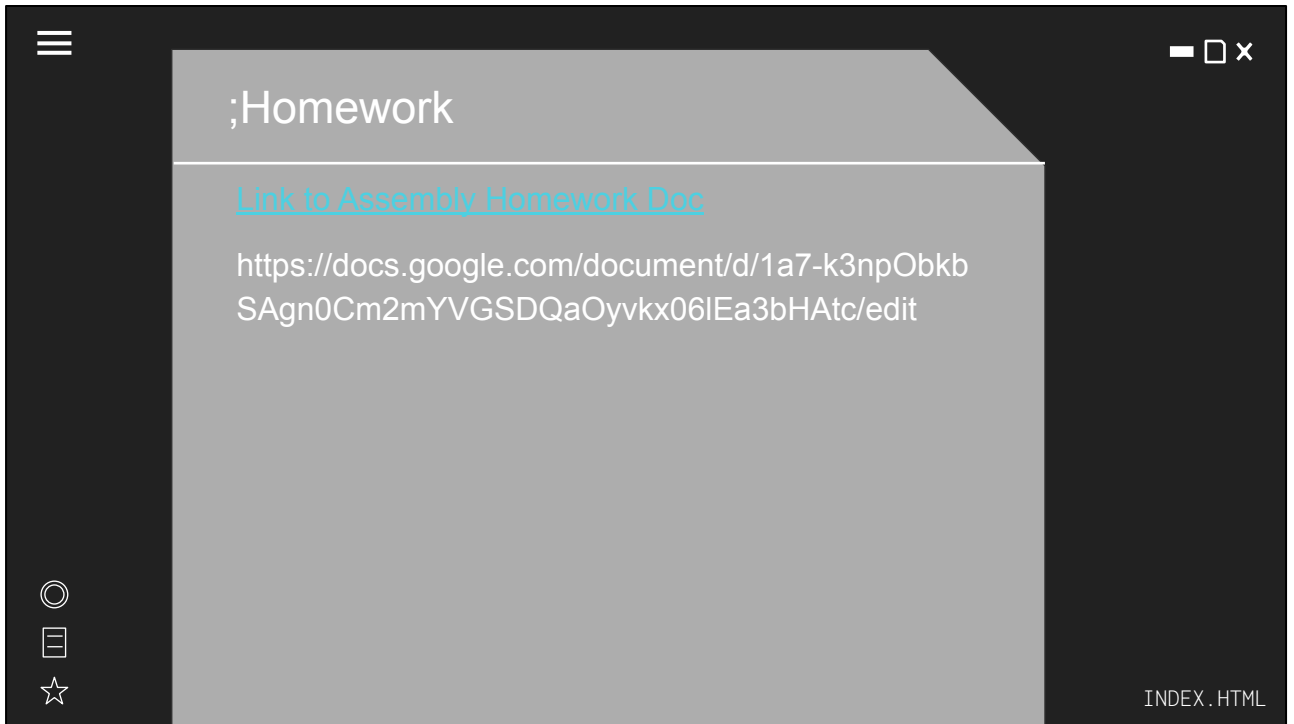
## ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

INDEX.HTML

Taylor + Amanda

This program only handles the sum as a single digit ASCII value– if the sum is above 9, the program reads it as a different ASCII value for a punctuation mark  
Could add 48 instead of adding "0" for the same effect



Ed

<https://docs.google.com/document/d/1a7-k3npObkbSAgn0Cm2mYVGSDQaOyvkvx06lEa3bHAtc/edit>



;Learn more!



INDEX.HTML



## ;Repeating Code

Assembly language doesn't contain *for* or *while* loops.

How can we repeat code in Assembly?





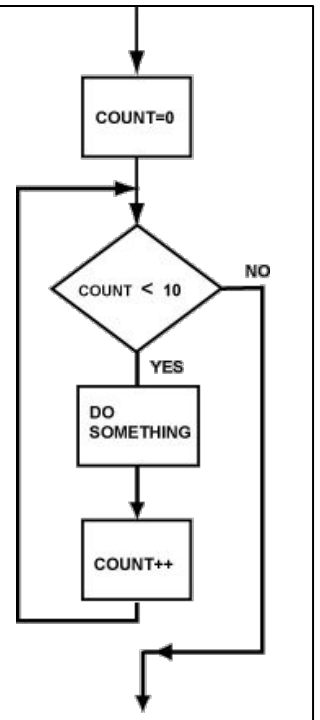
## ;Jumps

To repeat code, we use jumps:

JMP     always jumps

JE       jumps if two values are equal

JG       jumps if the first value is greater than the second







;Predict– what will happen if we run this code?

```
2 section .text
3     global _start      ;must be declared for use
4 _start:                ;tell linker entry point
5
6     ;move the data to registers
7     mov eax, 8
8     mov ebx, 4
9
10
11
12     cmp eax, ebx
13     jg a_larger
14
15     ; print message 1
16     mov edx, len1      ;message length
17     mov ecx, msg1      ;message to write
18     mov ebx, 1         ;file descriptor (stdout)
19     mov eax, 4         ;system call number (sys_write)
20     int 0x80           ;call kernel
21
22 a_larger:
23     ; print message 2
24     mov edx, len2      ;message length
25     mov ecx, msg2      ;message to write
26     mov ebx, 1         ;file descriptor (stdout)
27     mov eax, 4         ;system call number (sys_write)
28     int 0x80           ;call kernel
29
30     ;End connection to Linux system
31     mov eax, 1         ;system call number (sys_exit)
32     int 0x80           ;call kernel
33
34 section .data
35     msg1 db "B is larger than A"
36     len1 equ $ - msg1
37
38     msg2 db "A is larger than B"
39     len2 equ $ - msg2
```

Should print “A is larger than B”

```
section .text
    global _start      ;must be declared for using gcc
_start:                ;tell linker entry point
```

;move the data to registers

mov eax, 8

mov ebx, 4

cmp eax, ebx

jg a\_larger

; print message 1

mov edx, len1 ;message length

mov ecx, msg1 ;message to write

mov ebx, 1 ;file descriptor (stdout)

mov eax, 4 ;system call number (sys\_write)

int 0x80 ;call kernel

a\_larger:

; print message 2

```
mov    edx, len2    ;message length
mov    ecx, msg2    ;message to write
mov    ebx, 1       ;file descriptor (stdout)
mov    eax, 4       ;system call number (sys_write)
int     0x80        ;call kernel
```

```
;End connection to Linux system
```

```
mov    eax, 1       ;system call number (sys_exit)
int     0x80        ;call kernel
```

```
section .data
```

```
msg1 db "B is larger than A"
```

```
len1 equ $ - msg1
```

```
msg2 db "A is larger than B"
```

```
len2 equ $ - msg2
```



;What about now (changed row 9)?

```
2 section .text
3     global _start           ;must be declared for us
4     _start:                 ;tell linker entry point
5
6
7     ;move the data to registers
8     mov eax, 8
9     mov ebx, 9
10
11
12     cmp eax, ebx
13     jg a_larger
14
15     ; print message 1
16     mov edx, len1           ;message length
17     mov ecx, msg1           ;message to write
18     mov ebx, 1              ;file descriptor (stdout)
19     mov eax, 4              ;system call number (sys_write)
20     int 0x80                ;call kernel
21
22
23 a_larger:
24     ; print message 2
25     mov edx, len2           ;message length
26     mov ecx, msg2           ;message to write
27     mov ebx, 1              ;file descriptor (stdout)
28     mov eax, 4              ;system call number (sys_write)
29     int 0x80                ;call kernel
30
31     ;End connection to Linux system
32     mov eax, 1              ;system call number (sys_exit)
33     int 0x80                ;call kernel
34
35 section .data
36     msg1 db "B is larger than A"
37     len1 equ $ - msg1
38     msg2 db "A is larger than B"
39     len2 equ $ - msg2
```

Prints both messages, whoops. How do we fix it? Need to fix it by adding another jump after printing the first message, and adding a label in row 29 to jump to



## ;Fixed Code



```
2 section .text
3     global _start      ;must be declared for using gcc
4     _start:            ;tell linker entry point
5
6
7     ;move the data to registers
8     mov eax, 8
9     mov ebx, 9
10
11
12     cmp eax, ebx
13     jg a_larger
14
15     ; print message 1
16     mov edx, len1      ;message length
17     mov ecx, msg1      ;message to write
18     mov ebx, 1          ;file descriptor (stdout)
19     mov eax, 4          ;system call number (sys_write)
20     int 0x80            ;call kernel
21     jmp ending
22
23
24     a_larger:
25     ; print message 2
26     mov edx, len2      ;message length
27     mov ecx, msg2      ;message to write
28     mov ebx, 1          ;file descriptor (stdout)
29     mov eax, 4          ;system call number (sys_write)
30     int 0x80            ;call kernel
31
32     ending:
33     ;End connection to Linux system
34     mov eax, 1          ;system call number (sys_exit)
35     int 0x80            ;call kernel
36
37 section .data
38     msg1 db "B is larger than A"
39     len1 equ $ - msg1
40
41     msg2 db "A is larger than B"
42     len2 equ $ - msg2
```

;Program to compare 2 values

section.text

global \_start ;must be declared for using gcc  
\_start: ;tell linker entry point

;move the data to registers

mov eax, 8  
mov ebx, 9

cmp eax, ebx  
jg a\_larger

; print message 1

mov edx, len1 ;message length  
mov ecx, msg1 ;message to write  
mov ebx, 1 ;file descriptor (stdout)  
mov eax, 4 ;system call number (sys\_write)  
int 0x80 ;call kernel  
jmp ending

a\_larger:  
; print message 2

```
mov    edx, len2    ;message length
mov    ecx, msg2     ;message to write
mov    ebx, 1        ;file descriptor (stdout)
mov    eax, 4         ;system call number (sys_write)
int     0x80          ;call kernel
```

ending:

```
    ;End connection to Linux system
mov    eax, 1         ;system call number (sys_exit)
int     0x80          ;call kernel
```

section .data

```
msg1 db "B is larger than A"
len1 equ $ - msg1
```

```
msg2 db "A is larger than B"
len2 equ $ - msg2
```



## ;Sample Code: Add 2 numbers from user

```
1 section .text
2     global _start           ;must be declared for using gcc
3     _start:                 ;tell linker entry point
4
5     ;Prints message
6     mov     edx, len1       ;message length
7     mov     ecx, msg1       ;message to write
8     mov     ebx, 1          ;file descriptor (stdout)
9     mov     eax, 4          ;system call number (sys_write)
10    int     0x80            ;call kernel
11
12    ;Read and store the user input
13    mov     edx, 2           ;stores up to 2 bytes of info
14    mov     ecx, num1       ;variable to store the info as
15    mov     ebx, 0          ;file descriptor (stdin)
16    mov     eax, 3          ;system call (sys_read)
17    int     0x80            ;call kernel
18
19    ;print
20    mov     edx, len2       ;message length
21    mov     ecx, msg2       ;message to write
22    mov     ebx, 1          ;file descriptor (stdout)
23    mov     eax, 4          ;system call number (sys_write)
24    int     0x80            ;call kernel
25
26    ;Read and store the user input
27    mov     edx, 2           ;stores up to 2 bytes of info
28    mov     ecx, num2       ;variable to store the info as
29    mov     ebx, 0          ;file descriptor (stdin)
30    mov     eax, 3          ;system call (sys_read)
31    int     0x80            ;call kernel
32
33    ;move the data to registers
34    mov     eax, [num1]
35    sub     eax, '0'
36    mov     ebx, [num2]
37    sub     ebx, '0'
38
39    add     eax, ebx
40    add     eax, '0'
41    mov     [result], eax
42
43    ;print
44    mov     edx, len3       ;message length
45    mov     ecx, msg3       ;message to write
46    mov     ebx, 1          ;file descriptor (stdout)
47    mov     eax, 4          ;system call number (sys_write)
48    int     0x80            ;call kernel
```

```
section      .text
    global _start           ;must be declared for using gcc
_start:
    ;Prints message
    mov     edx, len1       ;message length
    mov     ecx, msg1       ;message to write
    mov     ebx, 1          ;file descriptor (stdout)
    mov     eax, 4          ;system call number (sys_write)
    int     0x80            ;call kernel

    ;Read and store the user input
    mov     edx, 2           ;stores up to 2 bytes of info
    mov     ecx, num1       ;variable to store the info as
    mov     ebx, 0          ;file descriptor (stdin)
    mov     eax, 3          ;system call (sys_read)
    int     0x80            ;call kernel

    ;print
    mov     edx, len2       ;message length
    mov     ecx, msg2       ;message to write
    mov     ebx, 1          ;file descriptor (stdout)
    mov     eax, 4          ;system call number (sys_write)
    int     0x80            ;call kernel

    ;Read and store the user input
    mov     edx, 2           ;stores up to 2 bytes of info
    mov     ecx, num2       ;variable to store the info as
    mov     ebx, 0          ;file descriptor (stdin)
    mov     eax, 3          ;system call (sys_read)
    int     0x80            ;call kernel
```

```

;move the data to registers
mov eax, [num1]
sub eax, '0'
mov ebx, [num2]
sub ebx, '0'

add eax, ebx
add eax, '0'
mov [result], eax

;print
mov     edx, len3      ;message length
mov     ecx, msg3      ;message to write
mov     ebx, 1         ;file descriptor (stdout)
mov     eax, 4         ;system call number (sys_write)
int     0x80           ;call kernel

;print
mov     edx, 1         ;message length
mov     ecx, result    ;message to write
mov     ebx, 1         ;file descriptor (stdout)
mov     eax, 4         ;system call number (sys_write)
int     0x80           ;call kernel

;End connection to Linux system
mov     eax, 1         ;system call number (sys_exit)
int     0x80           ;call kernel

section .data
msg1 db "What is the first number? "
len1 equ $ - msg1
msg2 db "What is your second number? "
len2 equ $ - msg2
msg3 db "The sum is "
len3 equ $ - msg3

; msg      db      'What is your name? ',0xa ;our string
; len      equ     $ - msg                      ;length of our string (found
by taking the location of the byte after msg [$] and subtracting the location
of msg, which gives the bytes in msg)

; msg1     db      'Hello, '                  ;our dear string
; len1     equ     $ - msg1                    ;length of our dear string

section .bss
;Uninitialized data
num1 resb 2
num2 resb 2
result resb 1

```



## ;Sample Code: Add 2 numbers from user

```
24
25 ;Read and store the user input
26 mov edx, 2 ;stores up to 2 bytes of info
27 mov ecx, num2 ;variable to store the info as
28 mov ebx, 0 ;file descriptor (stdin)
29 mov eax, 3 ;system call (sys_read)
30 int 0x80 ;call kernel
31
32 ;move the data to registers
33 mov eax, [num1]
34 sub eax, '0'
35 mov ebx, [num2]
36 sub ebx, '0'
37
38 add eax, ebx
39 add eax, '0'
40 mov [result], eax
41
42 ;print
43 mov edx, len3 ;message length
44 mov ecx, msg3 ;message to write
45 mov ebx, 1 ;file descriptor (stdout)
46 mov eax, 4 ;system call number (sys_write)
47 int 0x80 ;call kernel
48
49 ;print
50 mov edx, 1 ;message length
51 mov ecx, result ;message to write
52 mov ebx, 1 ;file descriptor (stdout)
53 mov eax, 4 ;system call number (sys_write)
54 int 0x80 ;call kernel
55
56
57 ;End connection to Linux system
58 mov eax, 1 ;system call number (sys_exit)
59 int 0x80 ;call kernel
60
61 section .data
62 msg1 db "What is the first number? "
63 len1 equ $ - msg1
64 msg2 db "What is your second number? "
65 len2 equ $ - msg2
66 msg3 db "The sum is "
67 len3 equ $ - msg3
68
69
70 section .bss ;Uninitialized data
71 num1 resb 2
72 num2 resb 2
73 result resb 1
```

```
section .text
global _start ;must be declared for using gcc
_start: ;tell linker entry point

;Prints message
mov edx, len1 ;message length
mov ecx, msg1 ;message to write
mov ebx, 1 ;file descriptor (stdout)
mov eax, 4 ;system call number (sys_write)
int 0x80 ;call kernel

;Read and store the user input
mov edx, 2 ;stores up to 2 bytes of info
mov ecx, num1 ;variable to store the info as
mov ebx, 0 ;file descriptor (stdin)
mov eax, 3 ;system call (sys_read)
int 0x80 ;call kernel

;print
mov edx, len2 ;message length
mov ecx, msg2 ;message to write
mov ebx, 1 ;file descriptor (stdout)
mov eax, 4 ;system call number (sys_write)
int 0x80 ;call kernel

;Read and store the user input
mov edx, 2 ;stores up to 2 bytes of info
mov ecx, num2 ;variable to store the info as
mov ebx, 0 ;file descriptor (stdin)
mov eax, 3 ;system call (sys_read)
int 0x80 ;call kernel
```



```

;move the data to registers
mov eax, [num1]
sub eax, '0'
mov ebx, [num2]
sub ebx, '0'

add eax, ebx
add eax, '0'
mov [result], eax

;print
mov     edx, len3      ;message length
mov     ecx, msg3      ;message to write
mov     ebx, 1         ;file descriptor (stdout)
mov     eax, 4         ;system call number (sys_write)
int     0x80           ;call kernel

;print
mov     edx, 1         ;message length
mov     ecx, result    ;message to write
mov     ebx, 1         ;file descriptor (stdout)
mov     eax, 4         ;system call number (sys_write)
int     0x80           ;call kernel

;End connection to Linux system
mov     eax, 1         ;system call number (sys_exit)
int     0x80           ;call kernel

section .data
msg1 db "What is the first number? "
len1 equ $ - msg1
msg2 db "What is your second number? "
len2 equ $ - msg2
msg3 db "The sum is "
len3 equ $ - msg3

; msg      db      'What is your name? ',0xa ;our string
; len      equ     $ - msg                    ;length of our string (found
by taking the location of the byte after msg [$] and subtracting the location
of msg, which gives the bytes in msg)

; msg1     db      'Hello, '                ;our dear string
; len1     equ     $ - msg1                  ;length of our dear string

section .bss
;Uninitialized data
num1 resb 2
num2 resb 2
result resb 1

```