;Assembly Language

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

;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…

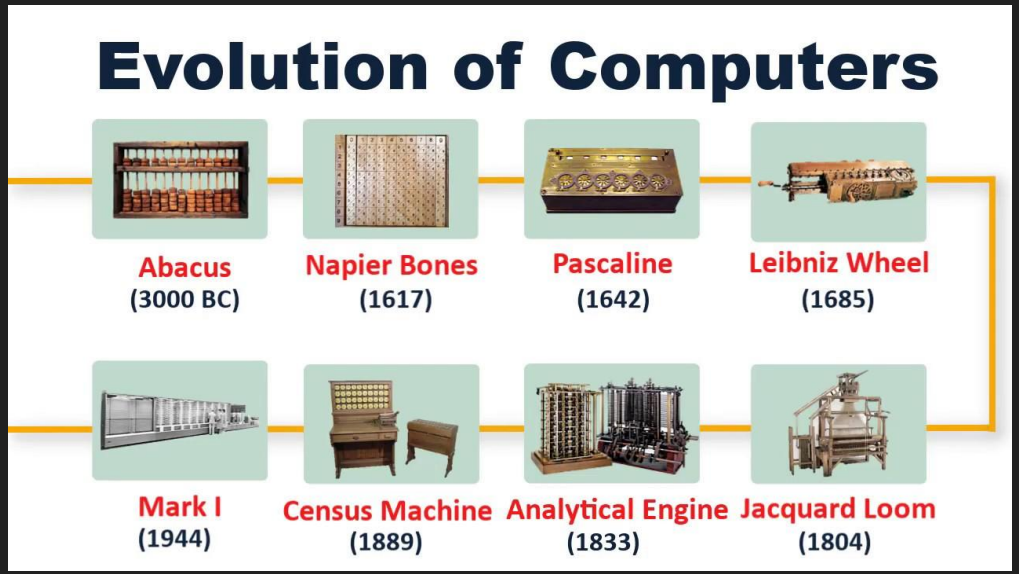
;Assembly: The code that tells your hardware what to do

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

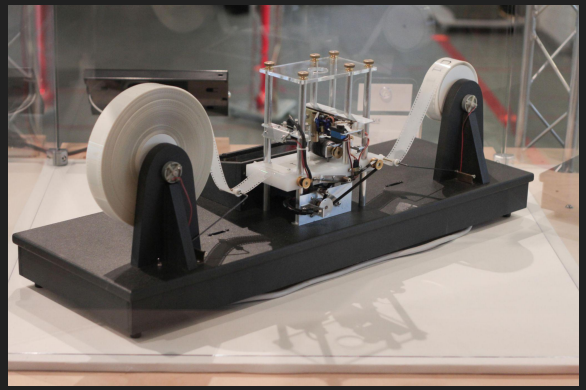
Mark 1: 5 tons, 816 cubic feet 1600 vacuum tubes, 5 horsepower (?!?!?)

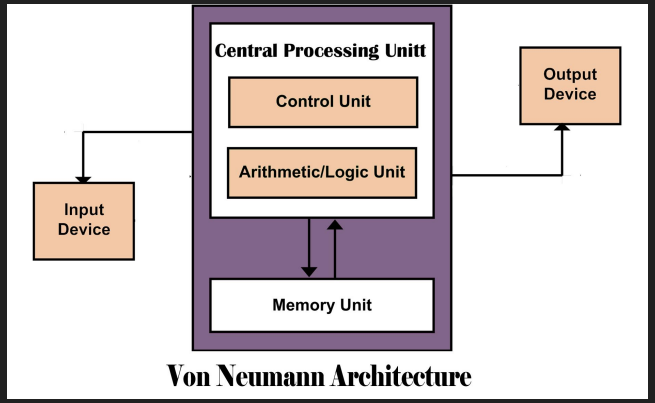
TRADIC: First transistorized computer, 800 transistors, only .13 horseys needed

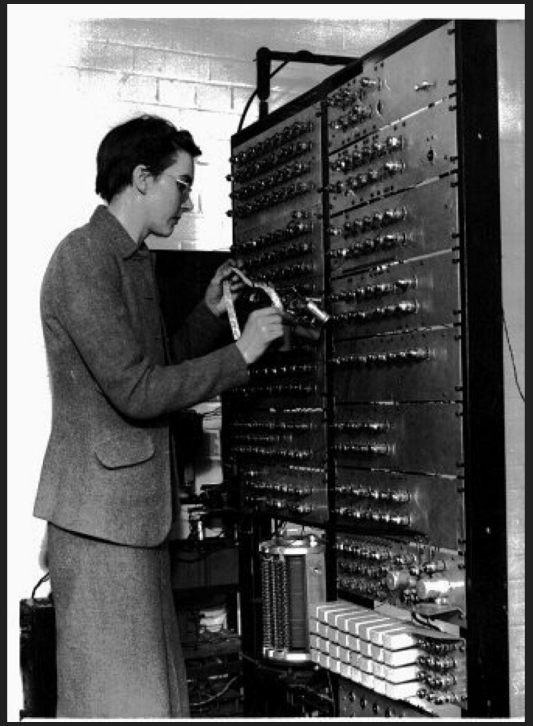
Modern CPU: ~290 million transistors and same # of horseys as TRADIC



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 × R → 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 ÷ M → cR.	Clear register, divide A by M, leave quotient in R and remainder in A.
13)	C → M_l.	
14)	C → M_r.	
15)	Cc → M_l.	If number in A ≥ 0 shift control to M_l.
16)	Cc → M_r.	
17)	A → 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

;Hello World

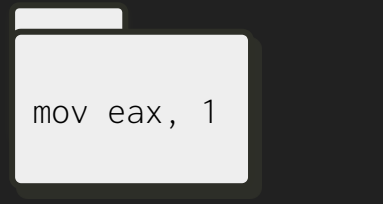
```
section .text
    global _start       ;must be declared for using gcc
_start:                     ;tell linker entry point
    mov edx, len    ;message length
    mov ecx, msg    ;message to write
    mov ebx, 1      ;file descriptor (stdout)
    mov eax, 4      ;system call number (sys_write)
    int 0x80        ;call kernel
    mov eax, 1      ;system call number (sys_exit)
    int 0x80        ;call kernel

section .data

msg db  'Hello, world!',0xa ;our dear string
len equ $ - msg         ;length of our dear string
```

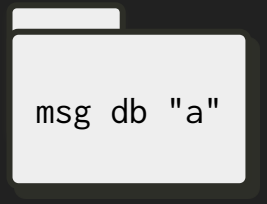
COMMENTS START WITH ;

;Assembly Sections



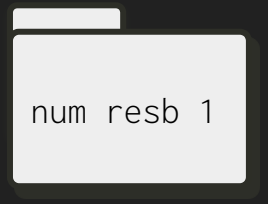
section .text

Contains the actual code (editor)



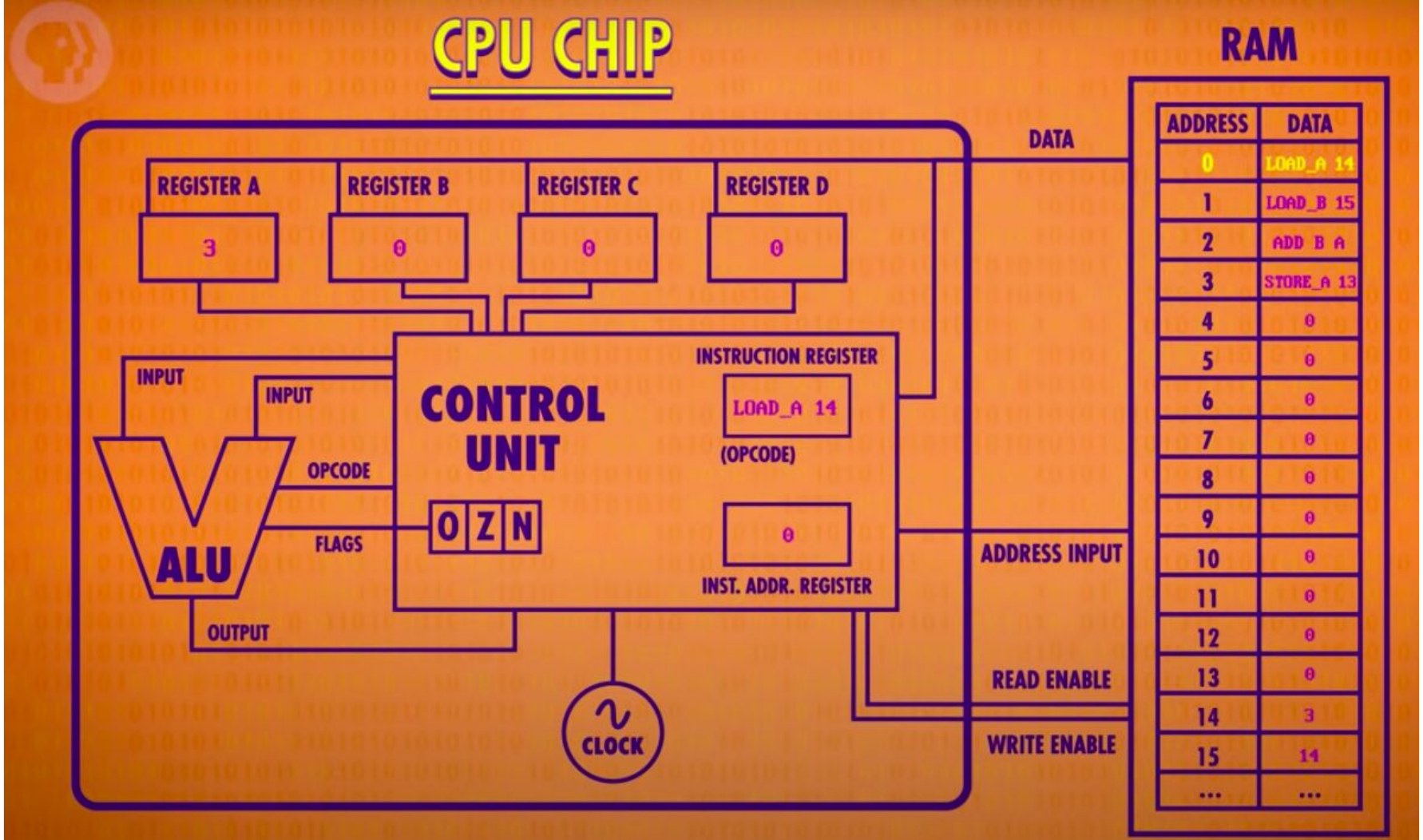
section .data

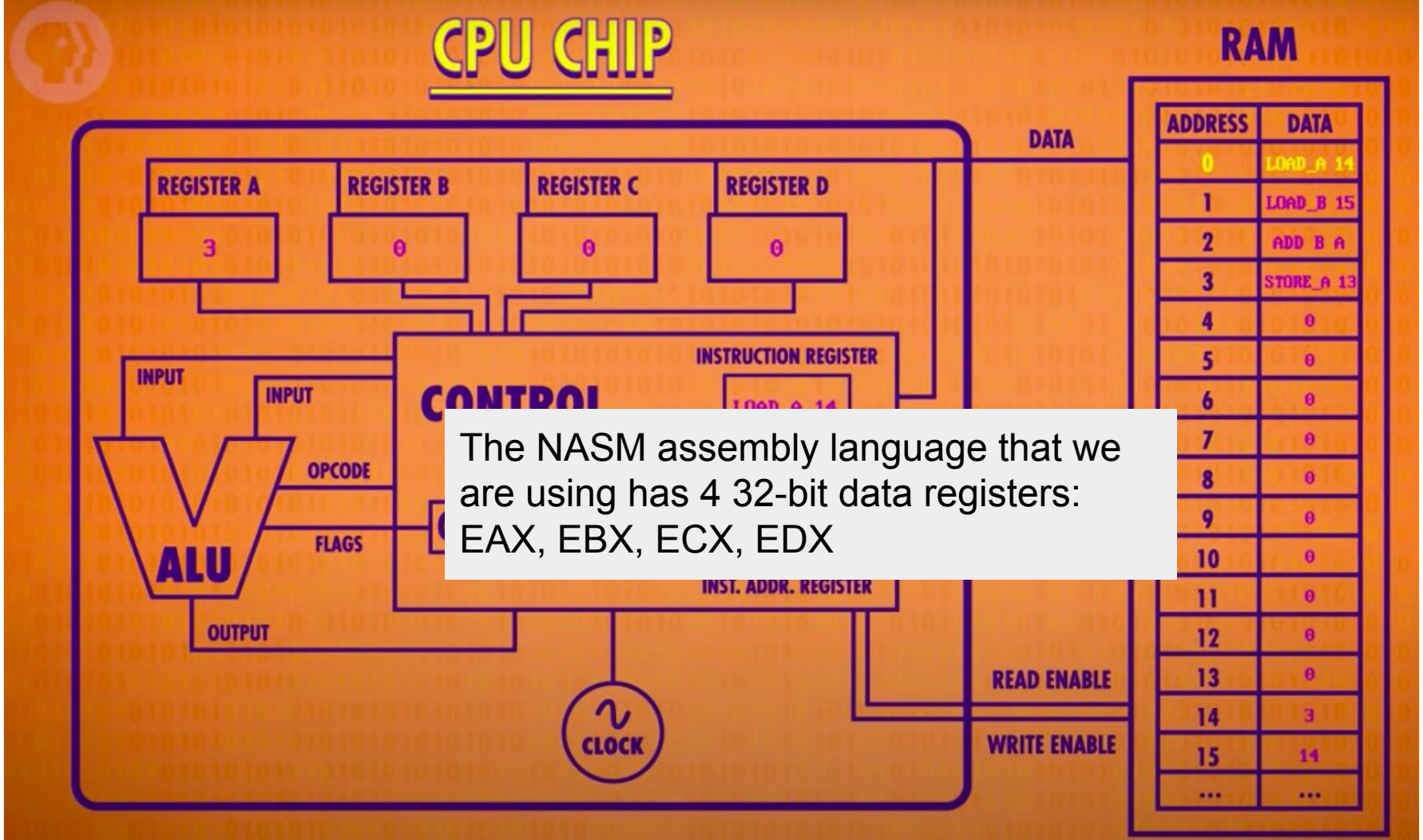
Declares **initialized** variables



section .bss

Declares **uninitialized** variables


CPU CHIP
REGISTER A
REGISTER B
REGISTER C
REGISTER D
3
0
0
0
INPUT
INPUT
ALU
OPCODE
FLAGS
OUTPUT
CONTROL UNIT
O Z N
INSTRUCTION REGISTER
LOAD_A 14
(OPCODE)
0
INST. ADDR. REGISTER
CLOCK
DATA
ADDRESS INPUT
READ ENABLE
WRITE ENABLE
RAM
ADDRESS
DATA
0
LOAD_A 14
1
LOAD_B 15
2
ADD B A
3
STORE_A 13
4
0
5
0
6
0
7
0
8
0
9
0
10
0
11
0
12
0
13
0
14
3
15
14
...
...



The NASM assembly language that we are using has 4 32-bit data registers: EAX, EBX, ECX, EDX

;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

;Assembly Variables

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

To declare a variable:

Variable-name declaration value

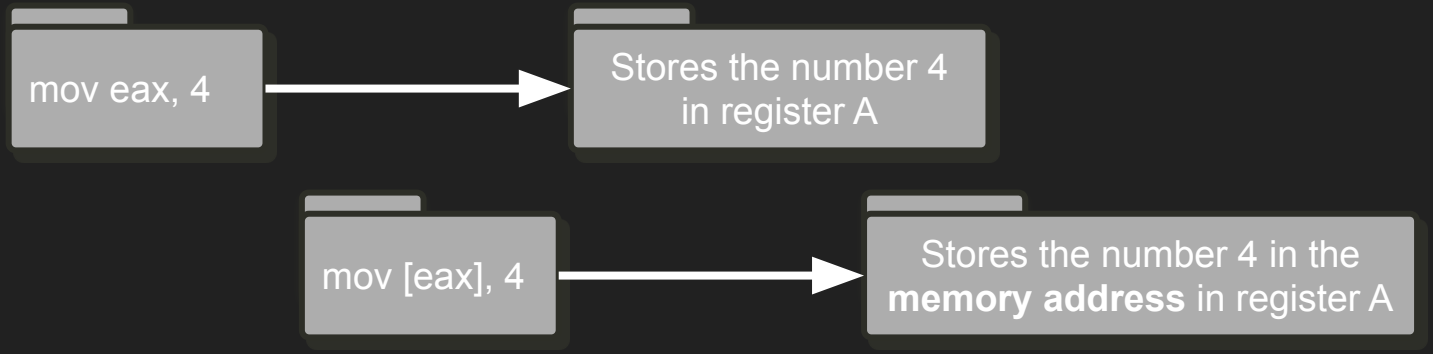
Ex:

```
message     db      “hello world”

counter     db      0
```

;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



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

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

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

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

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

Register A

3

Register B

1

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

Final

Register A

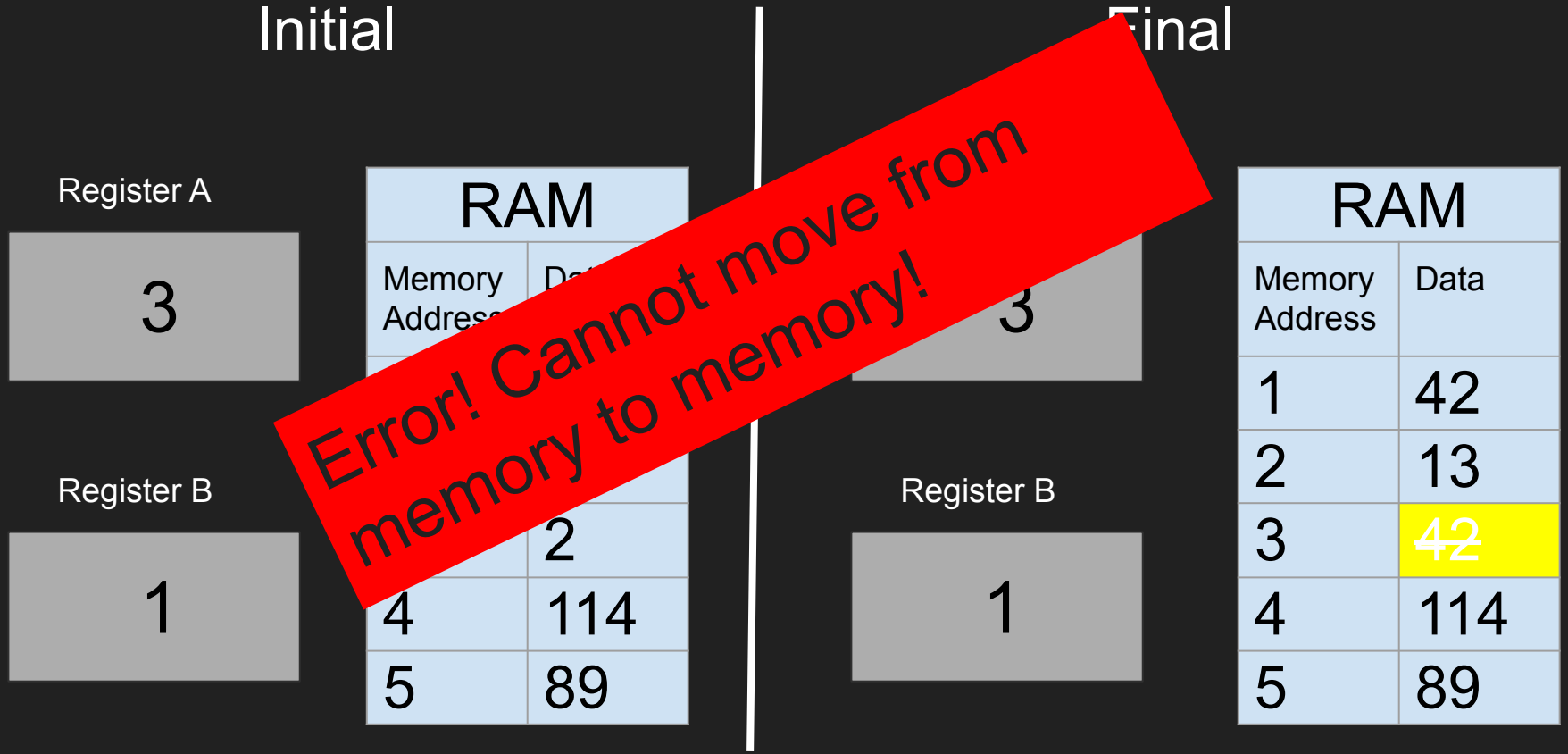
3

Register B

1

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

Code: mov [eax], [ebx]



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

Register A

5

Register B

1

RAM

Variable	Memory Address	Data
x		
y		2
	4	114
	5	89

Error! Cannot move from memory to memory!

Final

Register B

1

RAM

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

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

```
section .text
    global _start       ;must be declared for using gcc
_start:                     ;tell linker entry point
    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
    mov eax, 1      ;system call number (sys_exit)
    int 0x80        ;call kernel

section .data

msg db  'Hello, world!',0xa ;our dear string
len equ $ - msg         ;length of our dear string
```

;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

;Code-Along: Adding 2 Numbers

```
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' ;converts from number to ASCII value of
        number
    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?

;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]

;Homework

Link to Assembly Homework Doc

https://docs.google.com/document/d/1a7-k3npObkbSAgn0Cm2mYVGSDQaOyvkx06lEa3bHAtc/edit

;Learn more!

;Repeating Code

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

How can we repeat code in Assembly?

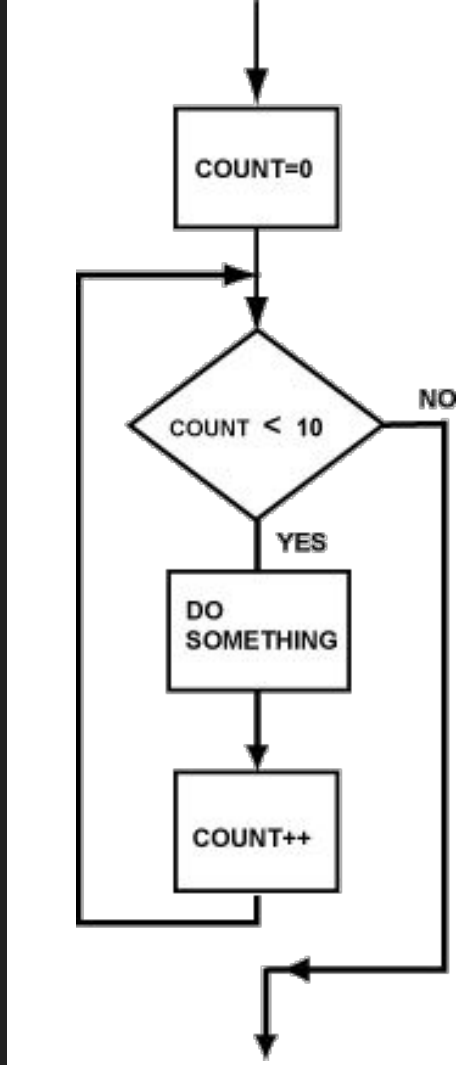
;Jumps

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?

```
section .text
    global _start       ;must be declared for us
_start:                 ;tell linker entry p


    ;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_wri
    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)?

```
section .text
    global _start       ;must be declared for us
_start:                 ;tell linker entry p


    ;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_wri
    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
```

;Fixed Code

```
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

```
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
```

;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
```

```
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
```