Memory in Assembly

Memory

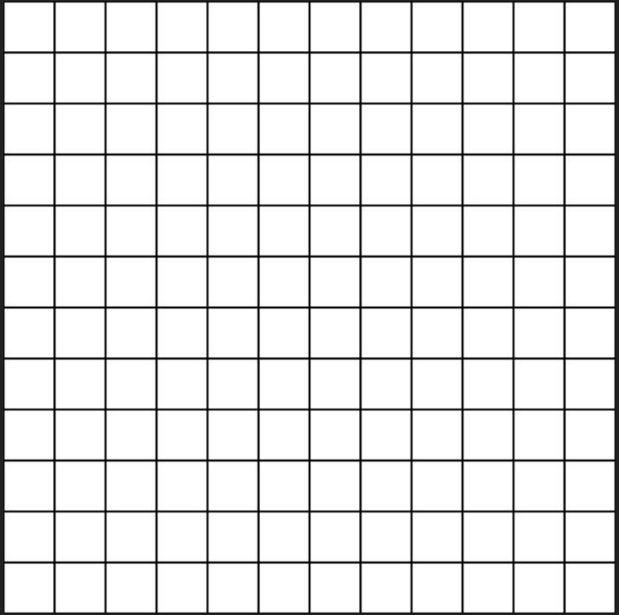
- So far, we’ve been mostly using the processor’s registers to store data
- In lab, we are going to explore the stack and memory
- Today we’ll talk more about addressing and accessing memory

Memory on our machines

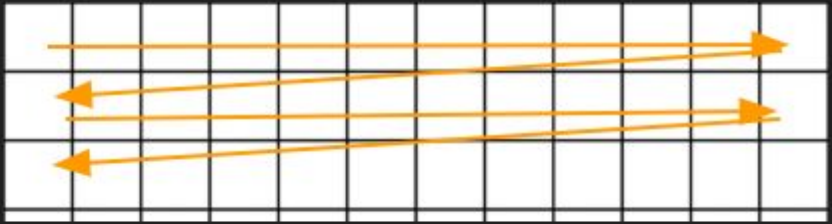
- The memory in our machines stores data so we can recall it later
- This occurs at several different levels
  - Networked drive (or cloud storage)
  - Hard drive
  - Dynamic memory
  - Cache
- For now, we can think of memory as a giant linear array.

Linear array of memory

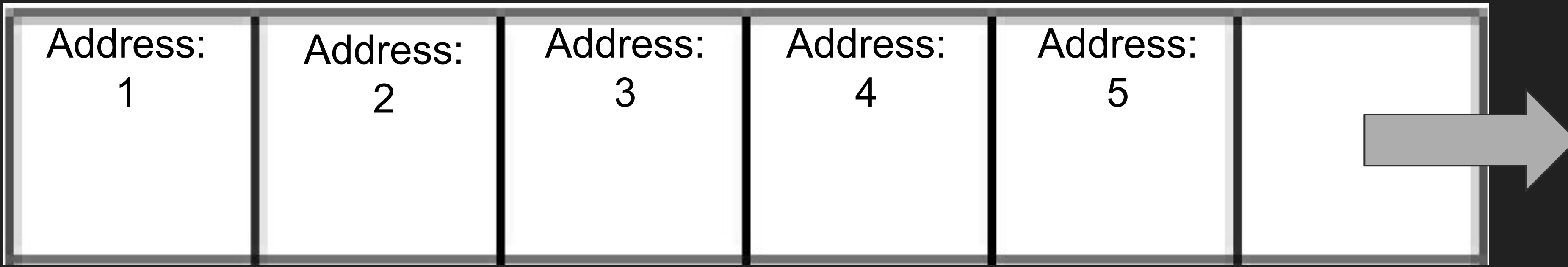
- Each ‘box’ here we will say is 1 byte of memory
  - (1 byte = 8 bits on most systems)
- Depending on the data we store, we will need 1 byte, 2 bytes, 4 bytes, etc. of memory



Linear array of memory



- Visually I have organized memory in a grid, but memory is really a linear array as depicted below.
  - There is 1 address after the other



Address: 1

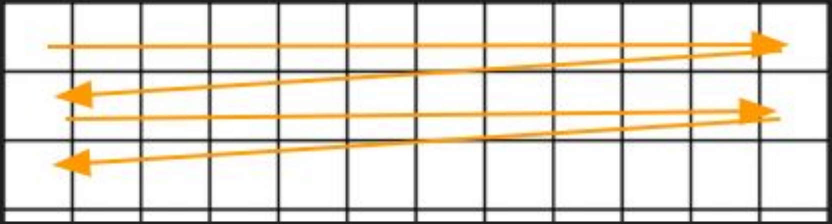
Address: 2

Address: 3

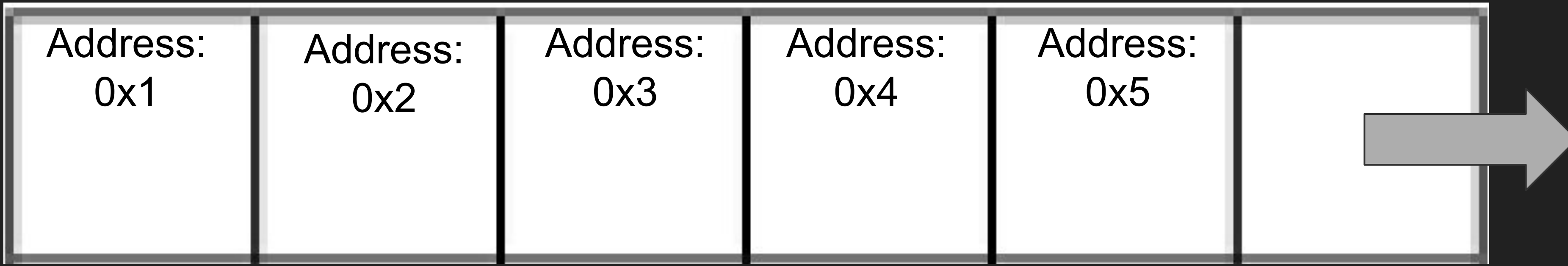
Address: 4

Address: 5

Linear array of memory



- Visually I have organized memory in a grid, but memory is really a linear array as depicted below.
  - There is 1 address after the other
  - Because these addresses grow large, typically we represent them in hexadecimal (16-base number system)
    - (https://www.rapidtables.com/convert/number/hex-to-decimal.html)



Address: 0x1

Address: 0x2

Address: 0x3

Address: 0x4

Address: 0x5

Remember: “Everything is a number”

Data Type	Suffix	Bytes	Range (unsigned)
`char`	**`b`**	1	0 to 255
`short int`	**`w`**	2	0 to 65,535
`int`	**`l`**	4	0 to 4,294,967,295
`long int`	**`q`**	8	0 to 18,446,744,073,709,551,615

Addressing memory

- Address granularity: **bytes**
- Suppose we are looking at a chunk of memory
- First address we see: 0x41F00 (in hexadecimal)
- This diagram: each row shows 8 bytes (aka one quadword = 64 bits)

…

0×41F00	*00*	*01*	*02*	*03*	*04*	*05*	*06*	*07*
0×41F08	*08*	*09*	*0A*	*0B*	*0C*	*0D*	*0E*	*0F*
0×41F10	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*
0×41F18	*18*	*19*	*1A*	*1B*	*1C*	*1D*	*1E*	*1F*
0×41F20	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*
0×41F28	*28*	*29*	*2A*	*2B*	*2C*	*2D*	*2E*	*2F*
0×41F30	*30*	*31*	*32*	*33*	*34*	*35*	*36*	*37*
0×41F38	*38*	*39*	*3A*	*3B*	*3C*	*3D*	*3E*	*3F*

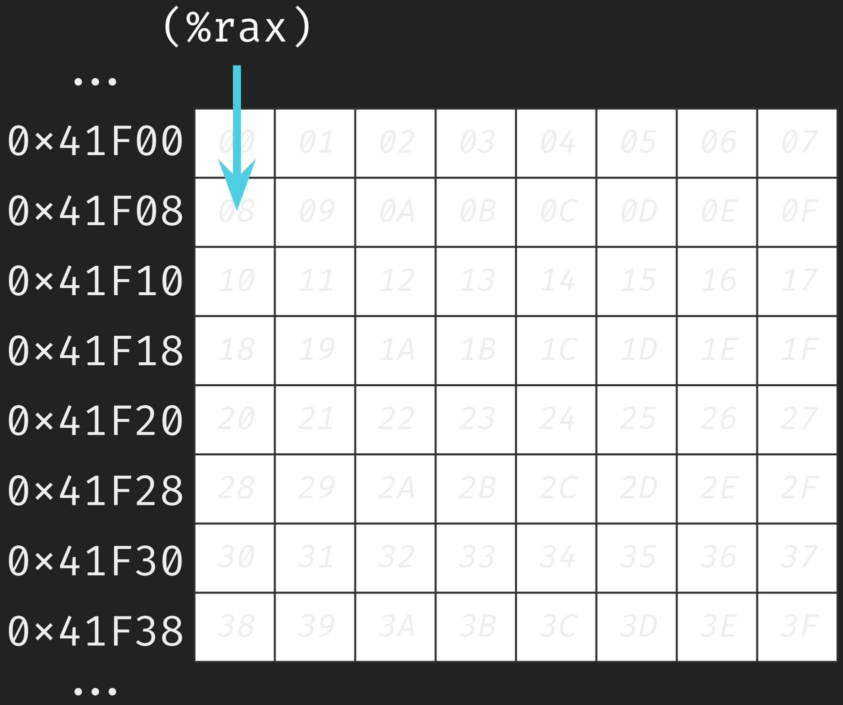
…

Addressing memory

```
movq $0×41F08, %rax
```

We move the address 0x41F08 into rax

(%rax) now points to the contents of the corresponding chunk of memory



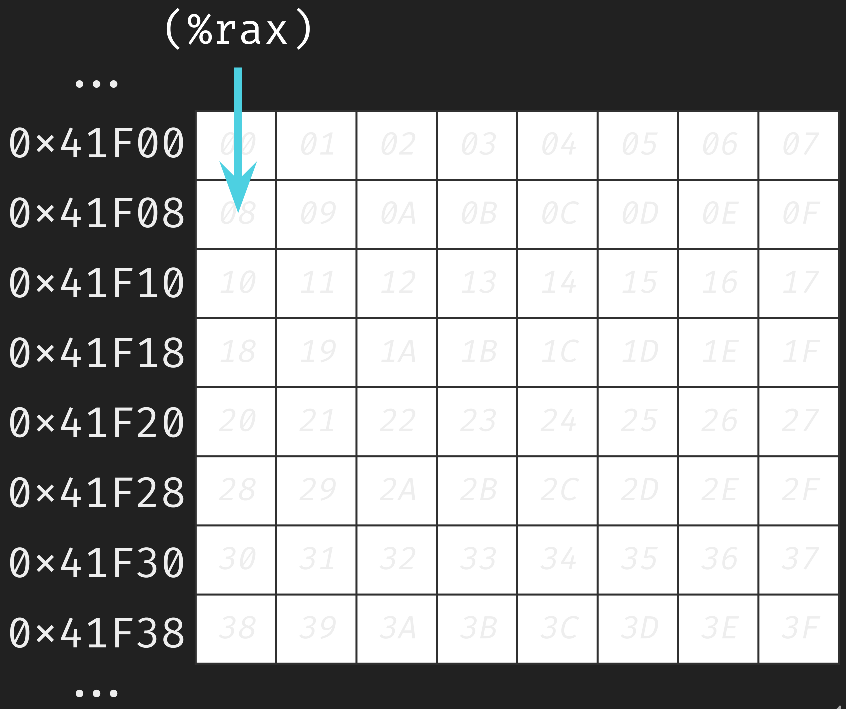
(%rax)

...								
0×41F00	00	01	02	03	04	05	06	07
0×41F08	08	09	0A	0B	0C	0D	0E	0F
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F
...								

Addressing memory

Offset addressing:

- We can point to addresses by adjusting the pointer register by an offset



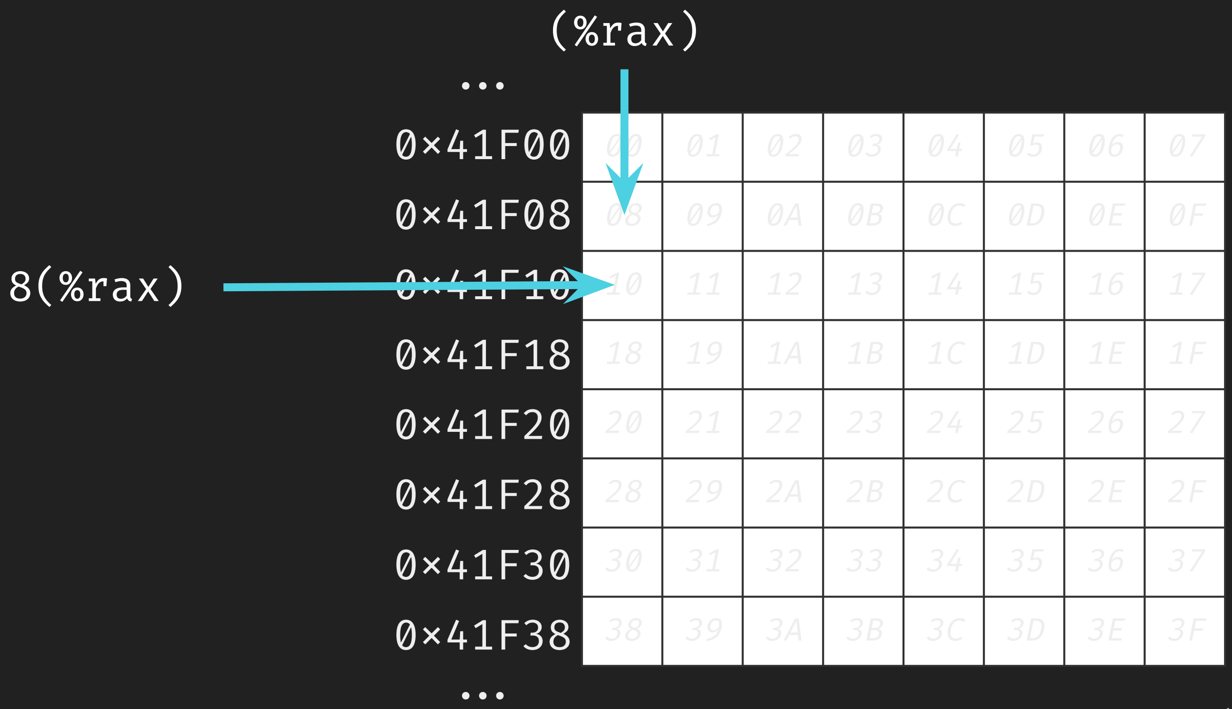
...

0x41F00	00	01	02	03	04	05	06	07
0x41F08	08	09	0A	0B	0C	0D	0E	0F
0x41F10	10	11	12	13	14	15	16	17
0x41F18	18	19	1A	1B	1C	1D	1E	1F
0x41F20	20	21	22	23	24	25	26	27
0x41F28	28	29	2A	2B	2C	2D	2E	2F
0x41F30	30	31	32	33	34	35	36	37
0x41F38	38	39	3A	3B	3C	3D	3E	3F

...

Addressing memory

Offset addressing



(%rax)

8(%rax)

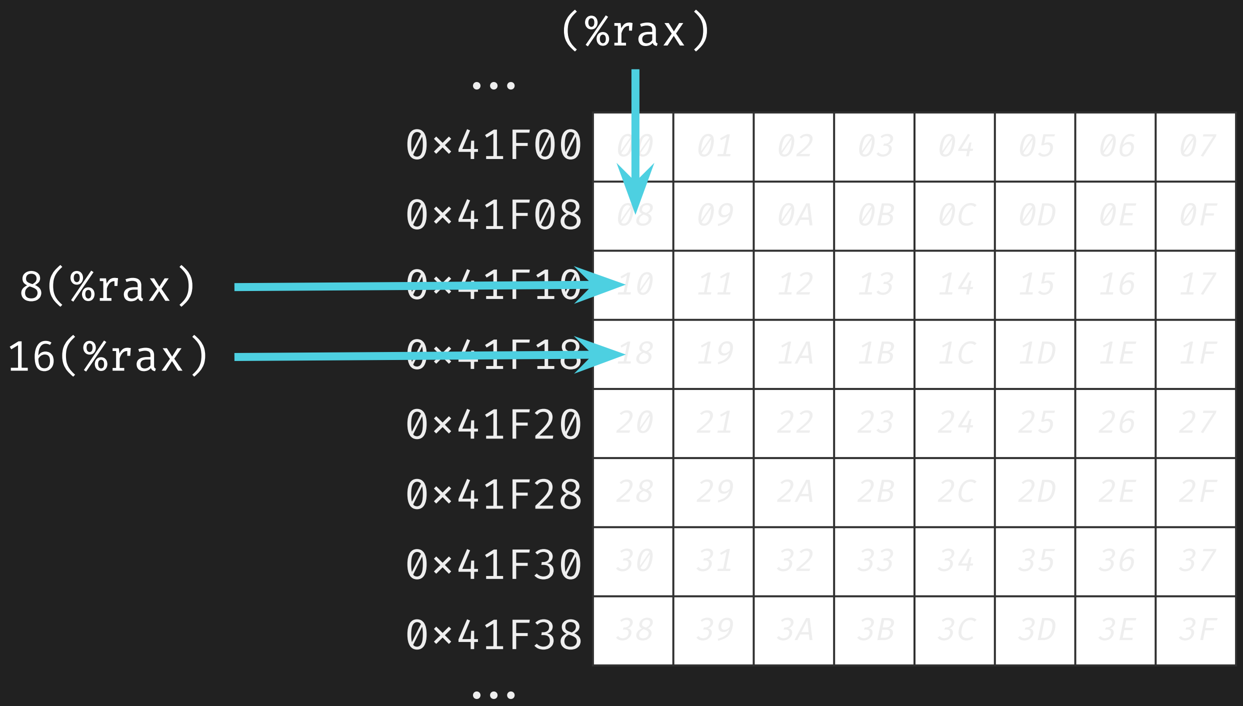
...

0×41F00	00	01	02	03	04	05	06	07
0×41F08	08	09	0A	0B	0C	0D	0E	0F
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

...

Addressing memory

Offset addressing



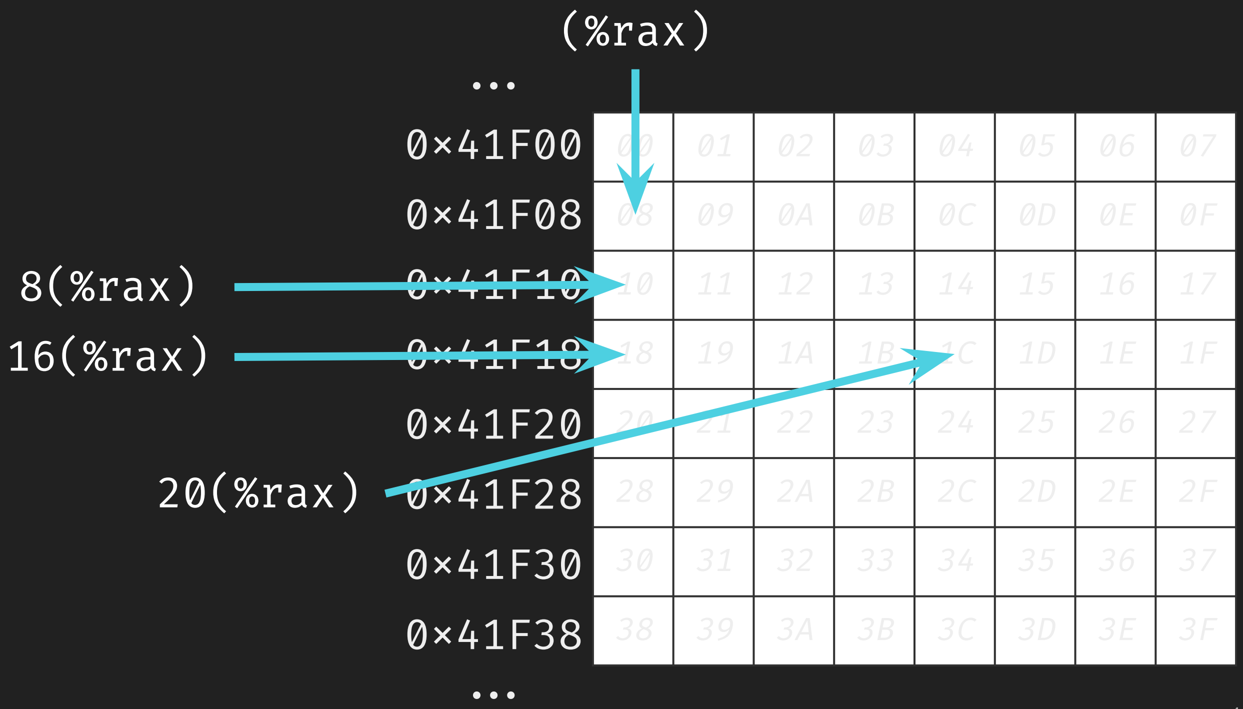
…

0×41F00	00	01	02	03	04	05	06	07
0×41F08	08	09	0A	0B	0C	0D	0E	0F
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

…

Addressing memory

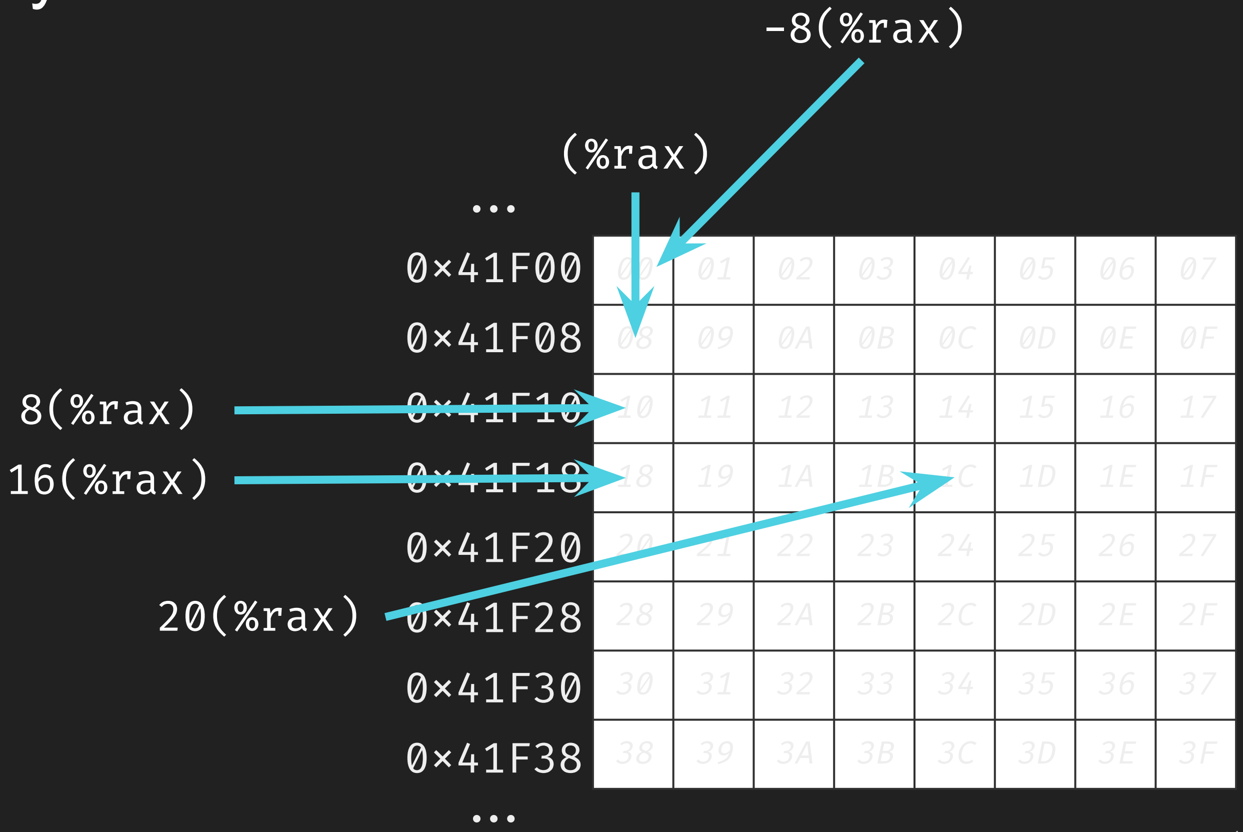
Offset addressing



0×41F00	00	01	02	03	04	05	06	07
0×41F08	08	09	0A	0B	0C	0D	0E	0F
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

Addressing memory

Offset addressing



-8(%rax)

(%rax)

8(%rax)

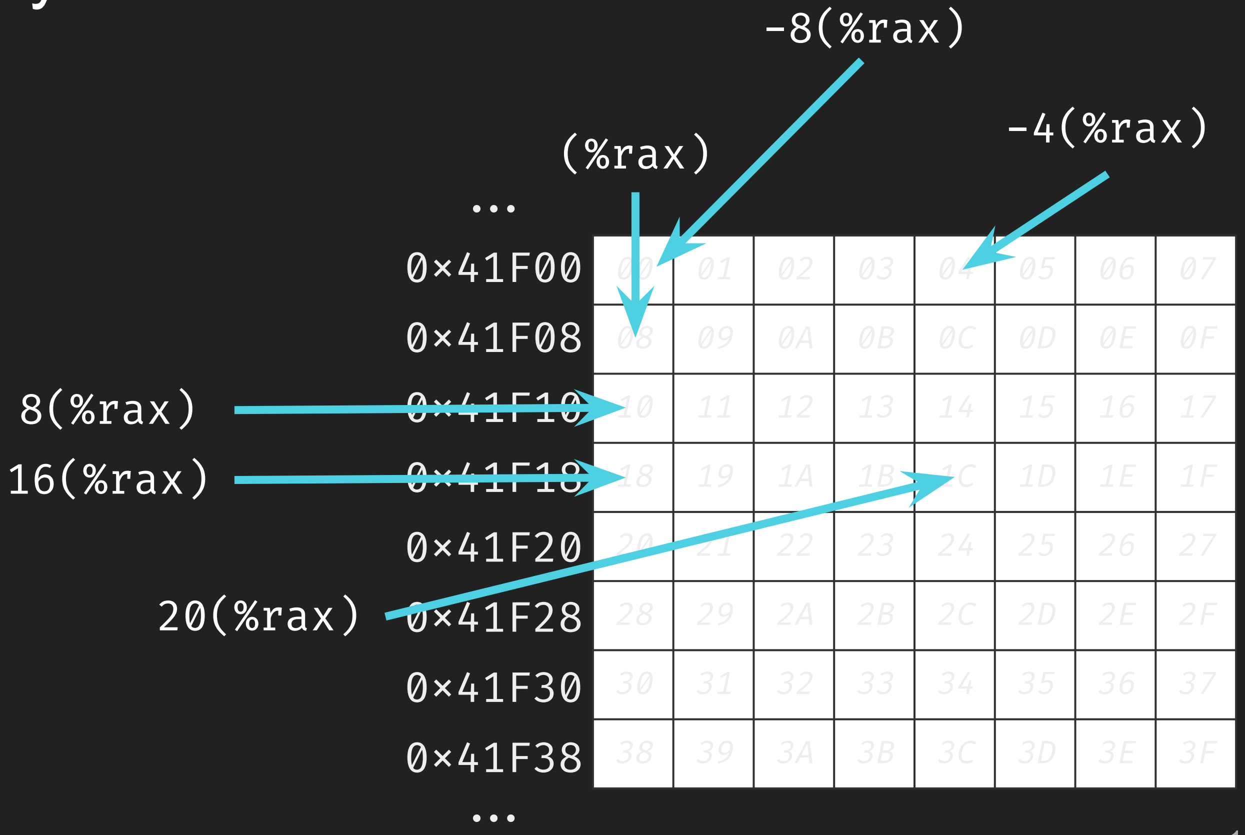
16(%rax)

20(%rax)

…								
0x41F00	00	01	02	03	04	05	06	07
0x41F08	08	09	0A	0B	0C	0D	0E	0F
0x41F10	10	11	12	13	14	15	16	17
0x41F18	18	19	1A	1B	1C	1D	1E	1F
0x41F20	20	21	22	23	24	25	26	27
0x41F28	28	29	2A	2B	2C	2D	2E	2F
0x41F30	30	31	32	33	34	35	36	37
0x41F38	38	39	3A	3B	3C	3D	3E	3F
…								

Addressing memory

Offset addressing



-8(%rax)

(%rax)

-4(%rax)

8(%rax)

16(%rax)

20(%rax)

...

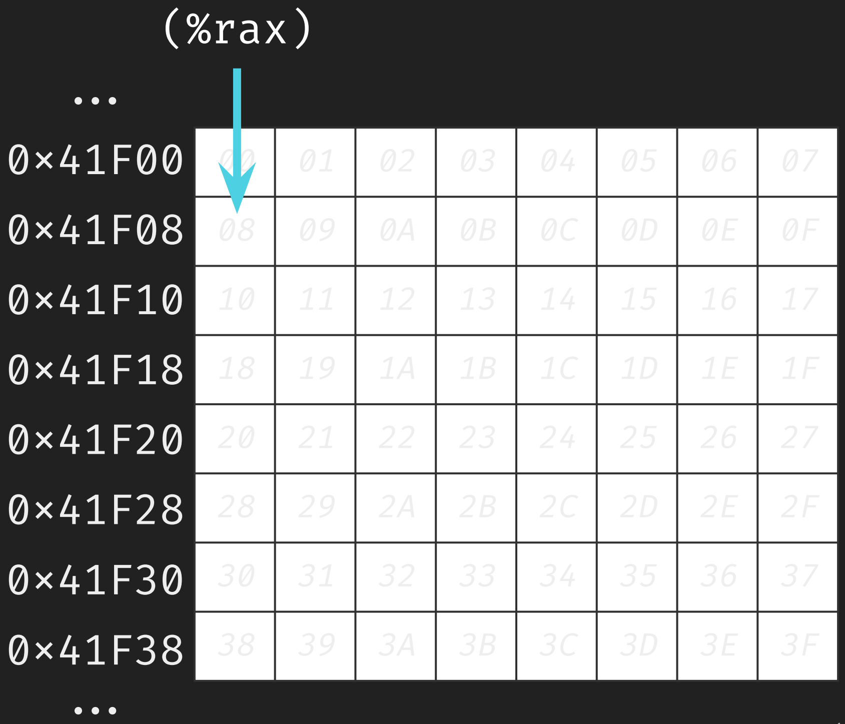
0x41F00	00	01	02	03	04	05	06	07
0x41F08	08	09	0A	0B	0C	0D	0E	0F
0x41F10	10	11	12	13	14	15	16	17
0x41F18	18	19	1A	1B	1C	1D	1E	1F
0x41F20	20	21	22	23	24	25	26	27
0x41F28	28	29	2A	2B	2C	2D	2E	2F
0x41F30	30	31	32	33	34	35	36	37
0x41F38	38	39	3A	3B	3C	3D	3E	3F

...

Addressing memory

```
movq $0×1020304050607080, (%rax)
```

What does this look like in memory?



(%rax)

...

0×41F00	00	01	02	03	04	05	06	07
0×41F08	08	09	0A	0B	0C	0D	0E	0F
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

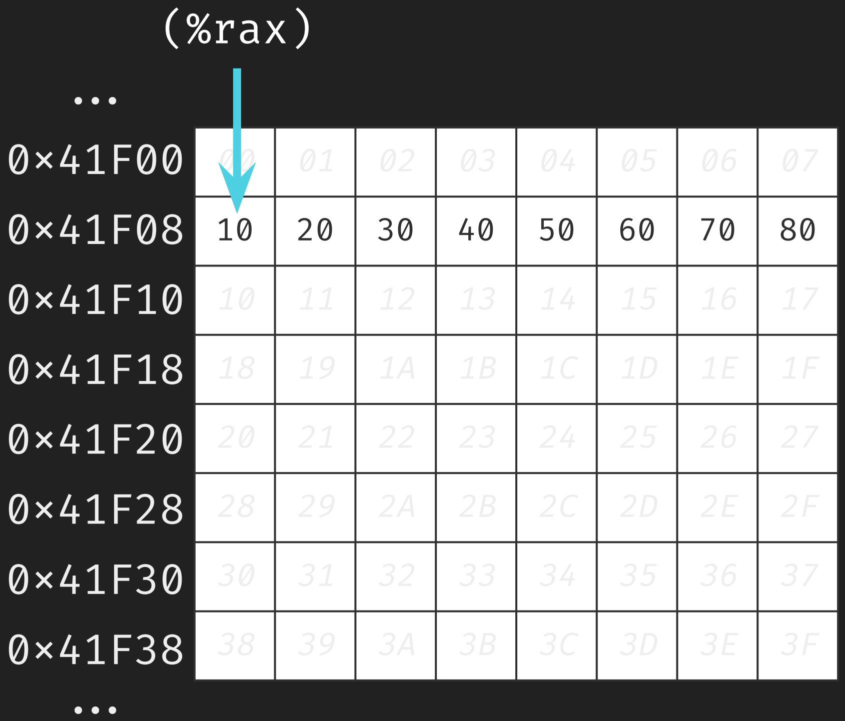
...

Addressing memory

```
movq $0×1020304050607080, (%rax)
```

What does this look like in memory?

Like this?



(%rax)

...

0×41F00	00	01	02	03	04	05	06	07
0×41F08	10	20	30	40	50	60	70	80
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

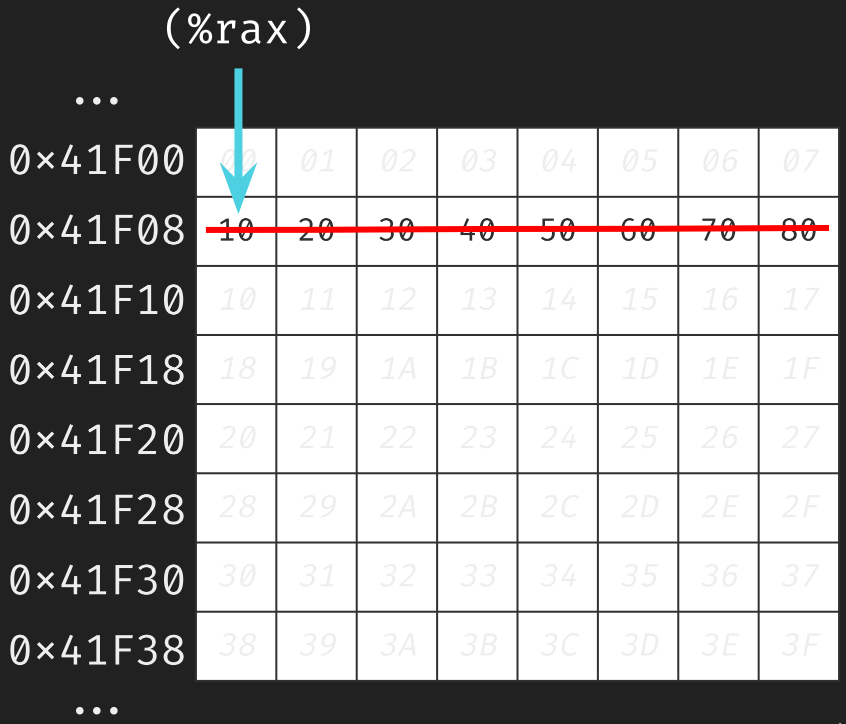
...

Addressing memory

```
movq $0×1020304050607080, (%rax)
```

What does this look like in memory?

Like this? **NO**



…

0×41F00	00	01	02	03	04	05	06	07
0×41F08	~~10~~	~~20~~	~~30~~	~~40~~	~~50~~	~~60~~	~~70~~	~~80~~
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

…

Addressing memory

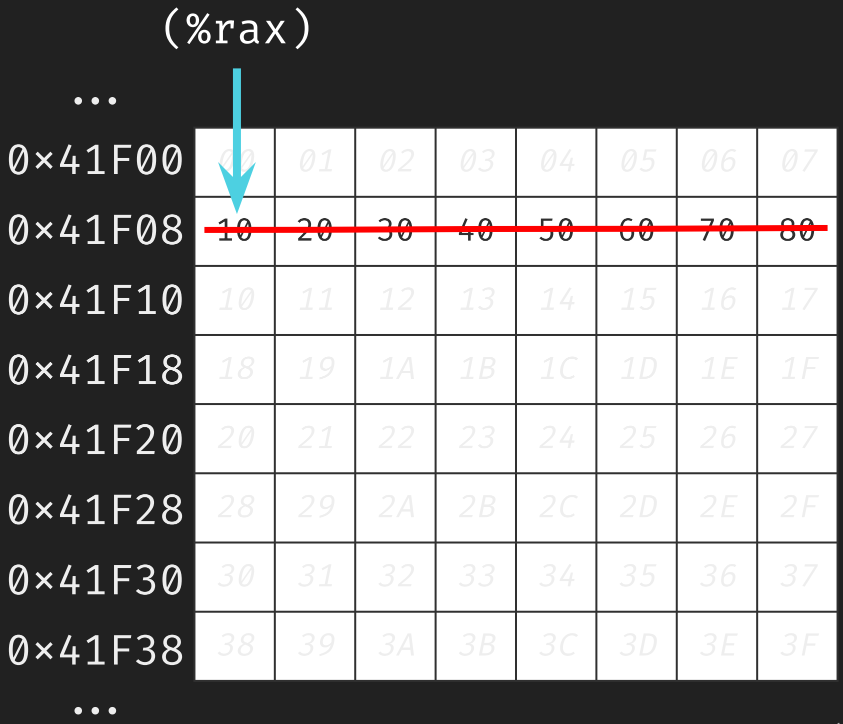
```
movq $0×1020304050607080, (%rax)
```

What does this look like in memory?

Like this? **NO**

→ x86 is *little-endian*: the less significant bytes are stored at lesser addresses

(end byte of the number, 0x80, is little)



(%rax)

...

0×41F00	00	01	02	03	04	05	06	07
0×41F08	~~10~~	~~20~~	~~30~~	~~40~~	~~50~~	~~60~~	~~70~~	~~80~~
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

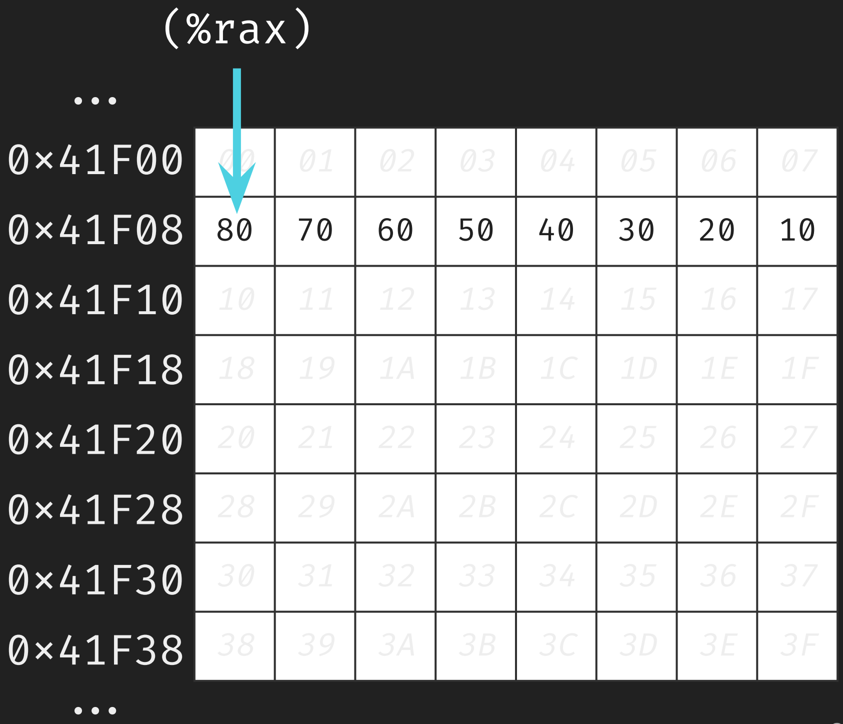
...

Addressing memory

```
movq $0×1020304050607080, (%rax)
```

What does this look like in memory?

Like this.



…								
0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F
…								

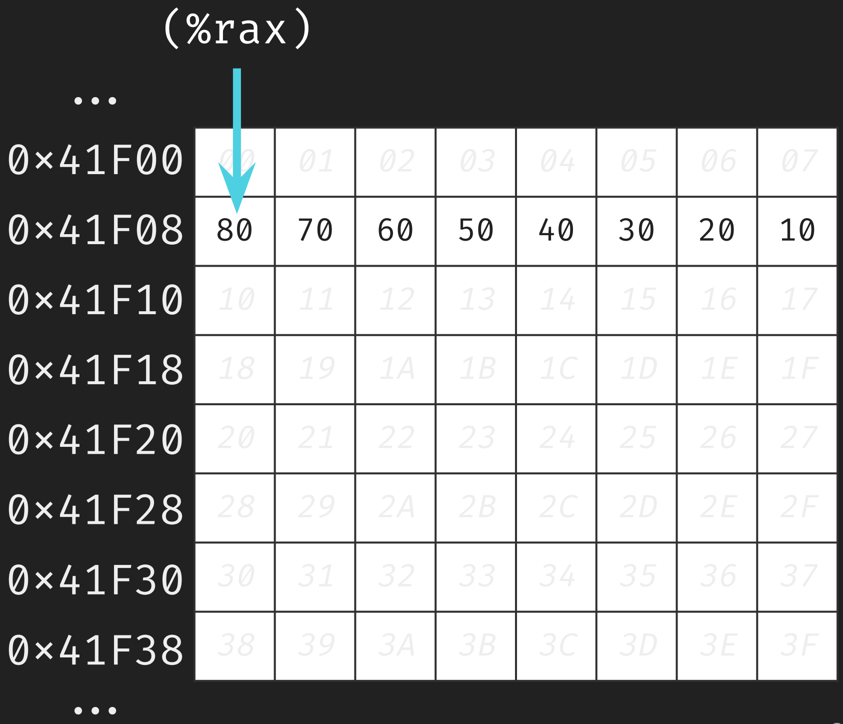
Addressing memory

```
movq (%rax), %r10
```

Copies the contents of the address pointed to by (%rax) to %r10

```
movq %rax, %r11
```

Copies the contents of %rax to %r11. Now (%rax) and (%r11) point to the same location.



...

0x41F00	00	01	02	03	04	05	06	07
0x41F08	80	70	60	50	40	30	20	10
0x41F10	10	11	12	13	14	15	16	17
0x41F18	18	19	1A	1B	1C	1D	1E	1F
0x41F20	20	21	22	23	24	25	26	27
0x41F28	28	29	2A	2B	2C	2D	2E	2F
0x41F30	30	31	32	33	34	35	36	37
0x41F38	38	39	3A	3B	3C	3D	3E	3F

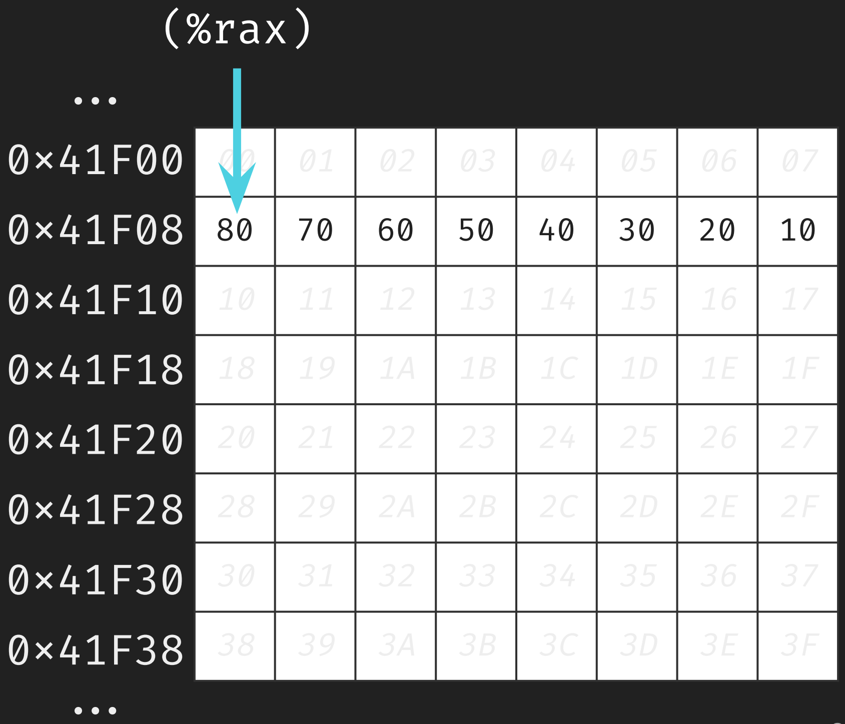
...

Addressing memory

```
movl (%rax), %ebx
```

What’s in %ebx?

How much we move is determined by operand sizes / suffixes



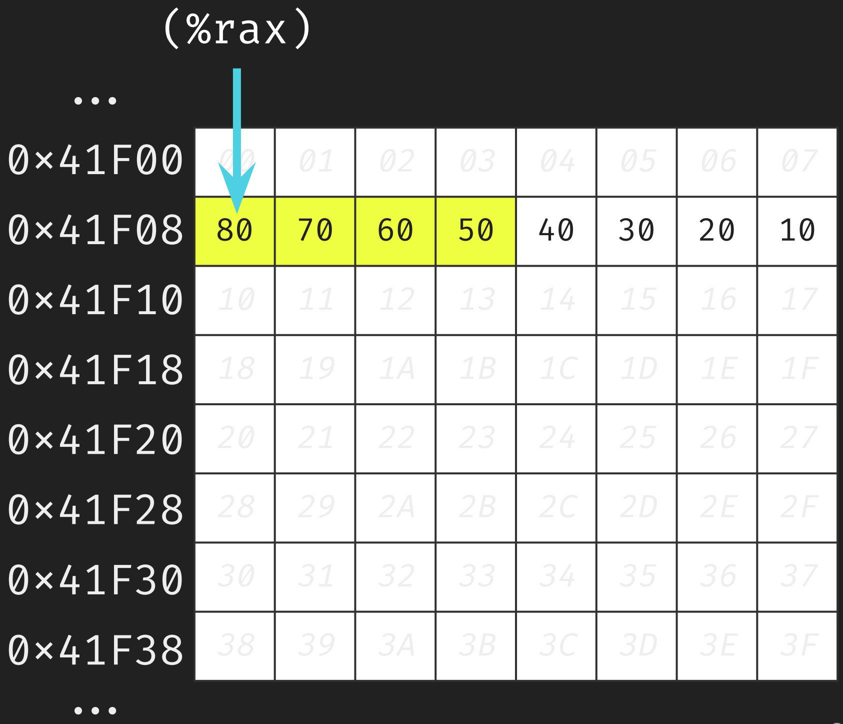
0x41F00	00	01	02	03	04	05	06	07
0x41F08	80	70	60	50	40	30	20	10
0x41F10	10	11	12	13	14	15	16	17
0x41F18	18	19	1A	1B	1C	1D	1E	1F
0x41F20	20	21	22	23	24	25	26	27
0x41F28	28	29	2A	2B	2C	2D	2E	2F
0x41F30	30	31	32	33	34	35	36	37
0x41F38	38	39	3A	3B	3C	3D	3E	3F

Addressing memory

```
movl (%rax), %ebx
```

What's in %ebx?

0x50607080



...

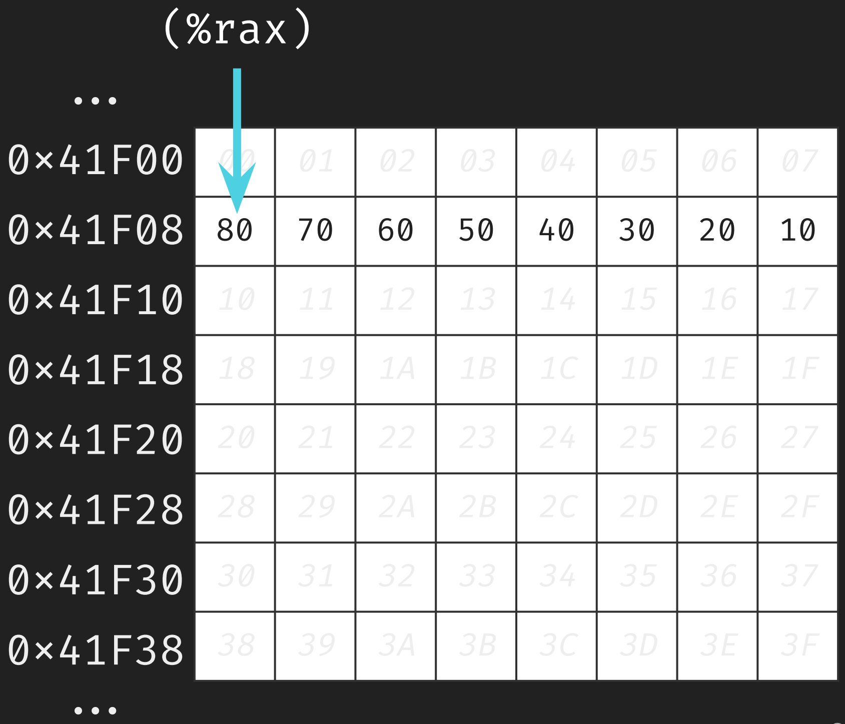
0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

...

Addressing memory

```
movw 4(%rax), %bx
```

What’s in %bx?



(%rax)

...

0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

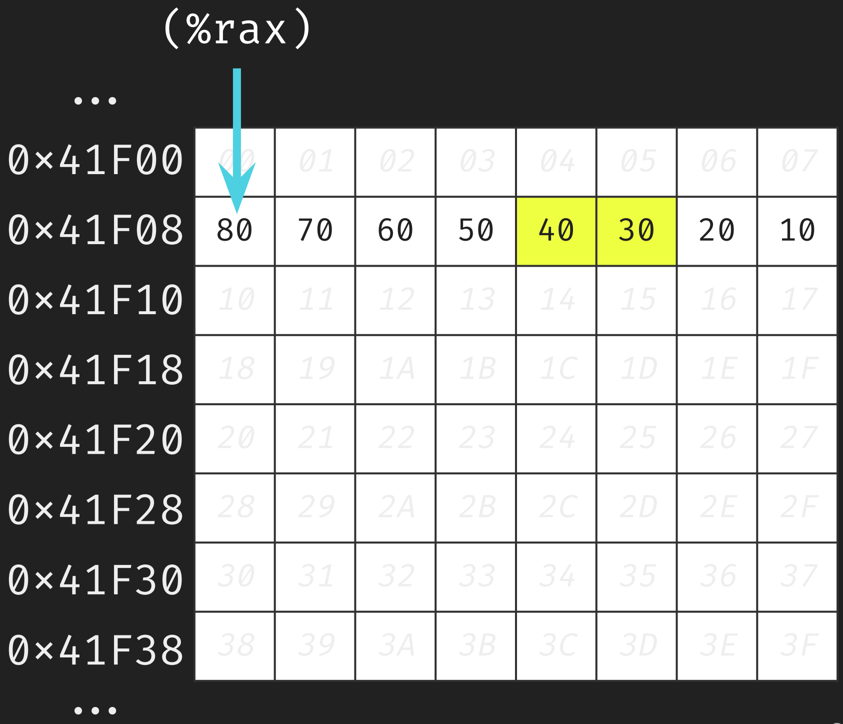
...

Addressing memory

```
movw 4(%rax), %bx
```

What's in %bx?

0x3040



…

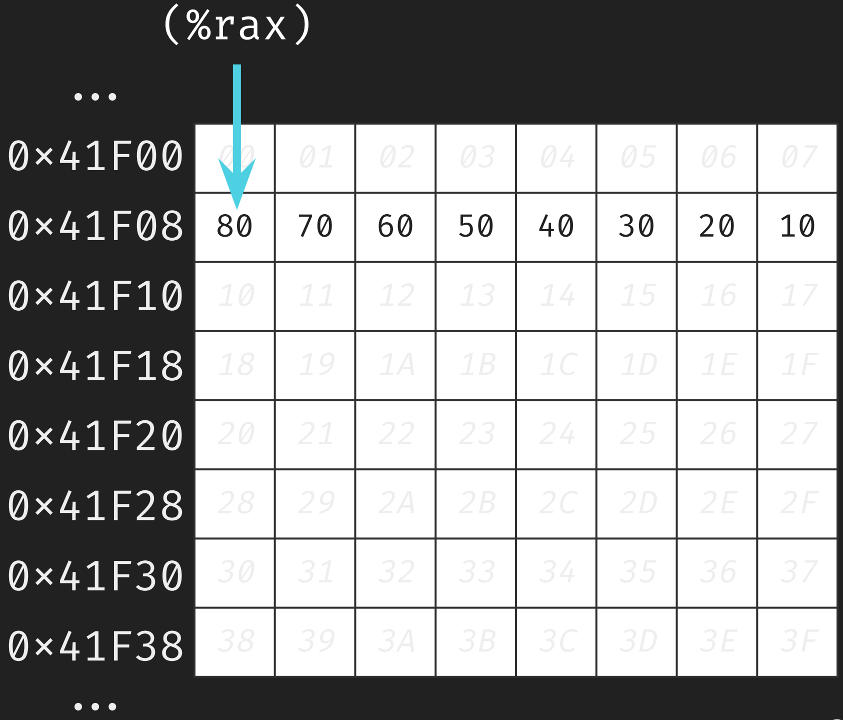
0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

…

Addressing memory

```
movb 6(%rax), %bl
```

What's in %bx?



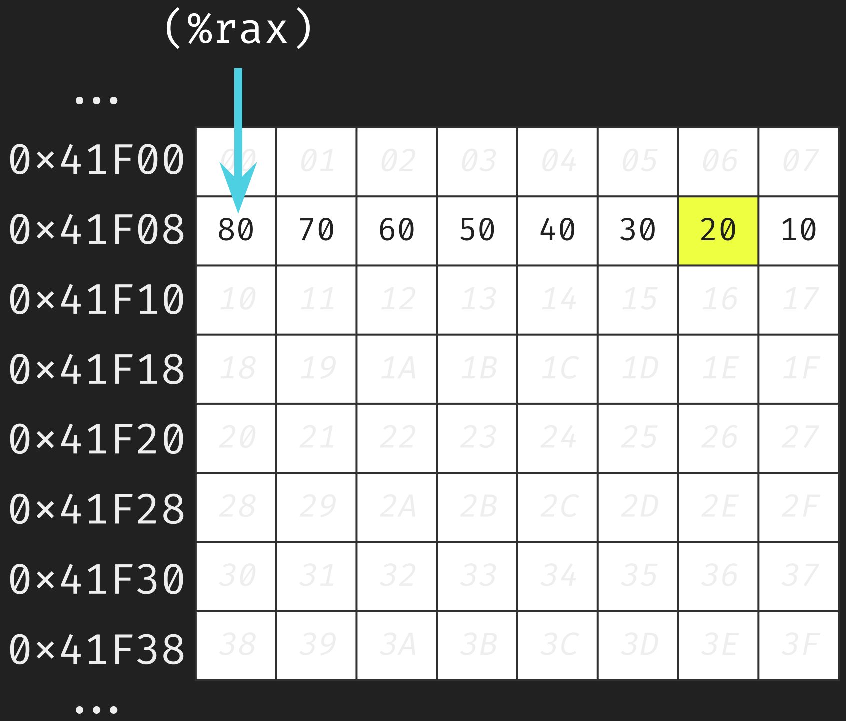
…								
0x41F00	00	01	02	03	04	05	06	07
0x41F08	80	70	60	50	40	30	20	10
0x41F10	10	11	12	13	14	15	16	17
0x41F18	18	19	1A	1B	1C	1D	1E	1F
0x41F20	20	21	22	23	24	25	26	27
0x41F28	28	29	2A	2B	2C	2D	2E	2F
0x41F30	30	31	32	33	34	35	36	37
0x41F38	38	39	3A	3B	3C	3D	3E	3F
…								

Addressing memory

```
movb 6(%rax), %bl
```

What’s in %bx?

0x3020



(%rax)

…

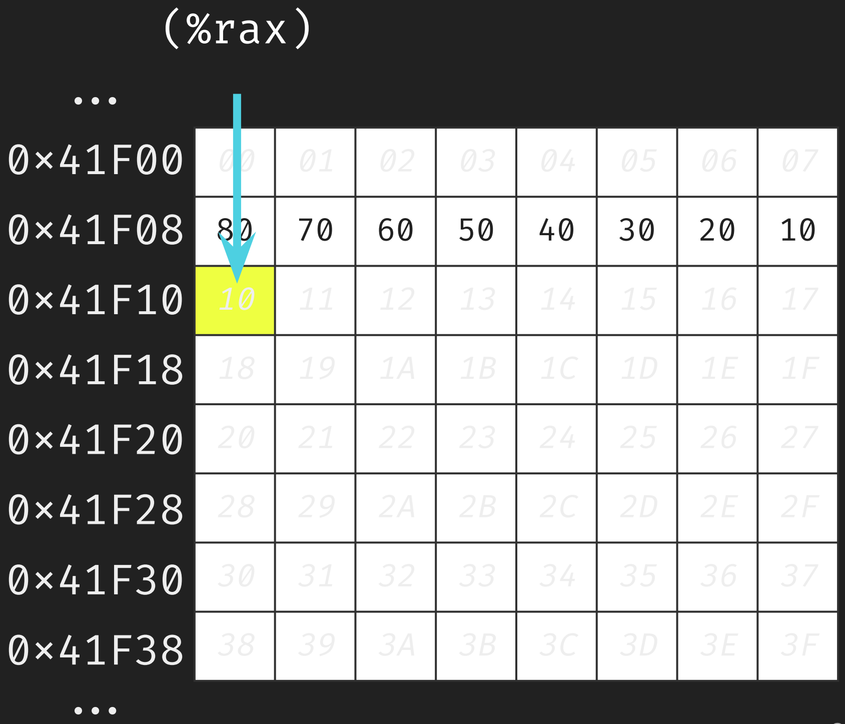
0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

…

Addressing memory

```
addq $8, %rax
```

Modifying %rax changes where it points



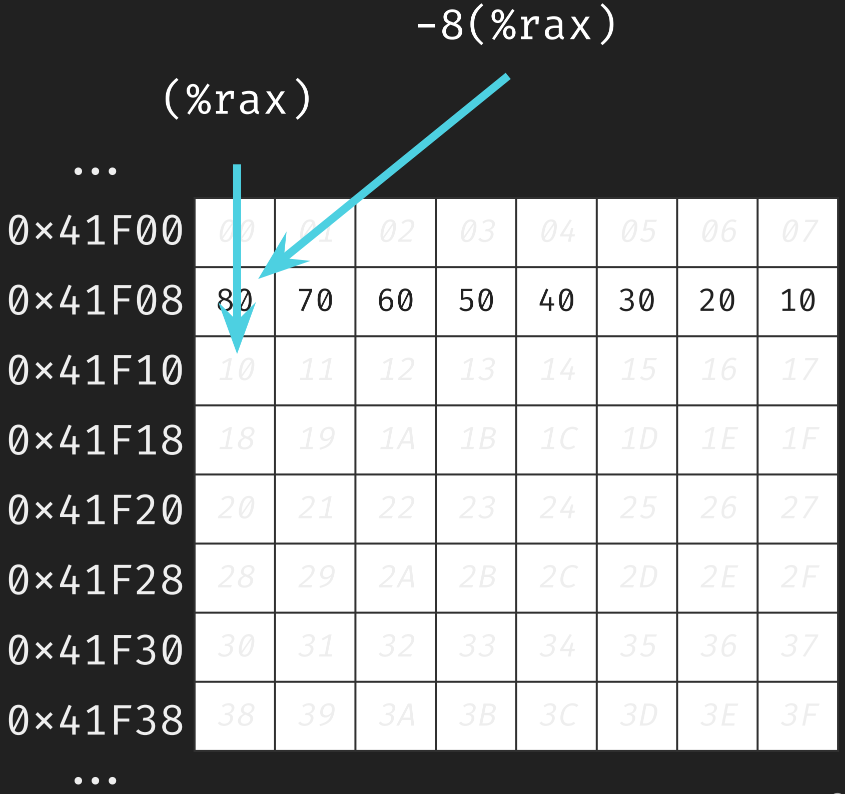
(%rax)

...								
0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F
...								

Addressing memory

```
addq $8, %rax
```

Modifying %rax changes where it points



…

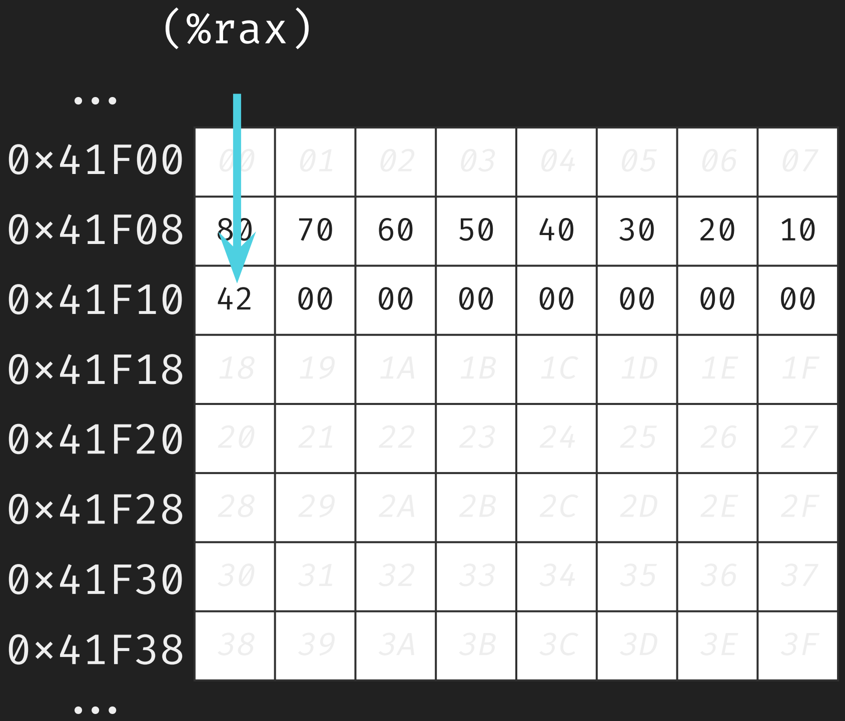
0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	10	11	12	13	14	15	16	17
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F

…

Addressing memory

```
addq $8, %rax
movq $0x42, (%rax)
```

Modifying %rax changes where it points



...								
0×41F00	00	01	02	03	04	05	06	07
0×41F08	80	70	60	50	40	30	20	10
0×41F10	42	00	00	00	00	00	00	00
0×41F18	18	19	1A	1B	1C	1D	1E	1F
0×41F20	20	21	22	23	24	25	26	27
0×41F28	28	29	2A	2B	2C	2D	2E	2F
0×41F30	30	31	32	33	34	35	36	37
0×41F38	38	39	3A	3B	3C	3D	3E	3F
...								

Addressing memory: full syntax

```
displacement(base, index, scale)
```

```
ADDRESS = base + (index * scale) + displacement
```

Mostly used for addressing arrays:

displacement: (immediate) offset / adjustment (e.g., -8, 8, 4, …)
base: (register) base pointer (%rax in previous examples)
index: (register) index of element
scale: (immediate) size of an element

Addressing memory: full syntax

```
displacement(base, index, scale)
```

```
ADDRESS = base + (index * scale) + displacement
```

Mostly used for addressing arrays:

displacement: (immediate) offset / adjustment (e.g., -8, 8, 4, …)
base: (register) base pointer (%rax in previous examples)
index: (register) element index
scale: (immediate) size of an element

Note:
`8(%rax)` is equivalent to `8(%rax, 0, 0)`

Addressing memory: full syntax

```
  mov $0x41F00, %rax

  mov $0, %rcx
  mov $0, %r10

loop:
  cmp $8, %rcx
  jge loop_end

  add (%rax, %rcx, 8), %r10
  inc %rcx
  jmp loop

loop_end:
```

...	
0x41F00	01
0x41F08	02
0x41F10	03
0x41F18	04
0x41F20	05
0x41F28	06
0x41F30	07
0x41F38	08
...	

What’s in %r10 after loop_end?