last time (1)

user and group IDs
- tracked per process
- set by program at login

superuser/root
- special user ID that ignores permission checks

Unix/POSIX standard permissions
- read/write/execute for one user+one group+others
- “owner” user can change permissions

POSIX access control lists (ACLs)
- flexible lists of user+groups and permissions

last time (2)

virtual memory

- program addresses = virtual
- machine addresses = physical

addresses divided into fixed size pages

- page size always power of two
- upper bits of addresses = page number (index)
- lower bits = page offset (location in page)

page table (virtual $\rightarrow$ physical mapping)

- row for each virtual page
- indicates physical page
- valid bit — is there any?

anonymous feedback (1)

"I think you cut off people during their questions too much. It would be nice if students could finish their questions and get them answered correctly."

anonymous feedback (2)

"I'm not sure if this applies to every single lab but at least for mine (330-445) the lab room feels insanely crowded. There often aren't enough chairs for everyone and one of the TAs told me that a lot of people just dont leave after their lab and stay for 2 or more lab sessions...I'm not really sure how this could be solved I just thought it was worth pointing out because the effect sort of compounds into later lab sessions since people who have a late lab and aren't able to finish in lab dont have as much time to work on it after lab class."

anonymous feedback (3)

“The in class example questions are really helpful and I would appreciate if you incorporated more of them somehow. Sometimes when it’s just content for a long time and no application/practice the information doesn’t click. I know it may not be applicable to all topics and that it may take up some class time but I guess one suggestion would be leaving us with a question or two at the end of lecture so we can work through it on our own time and you can give us the answers at the beginning of the next lecture or during OH.”

aside on sudo

should have explained what sudo is

utility system admin configures to allow some people to run things with extra permissions

usually prompts for password first

trick: because set-user-ID program, program with if statements

kernel “delegates” decision to the program

quiz Q1

A: change exception table — NO

- exception table is what hardware uses to run OS
- what OS runs is in kernel mode + doesn't know how to run normal function
- OS needs to call signal handler itself — hardware won't do it right

C: value of x on stack/register whne it runs

- stack: where Linux saves registers of interrupted function while signal handler running
- needs to be accessible because it would be used for the printf call

quiz Q2

A: signal handler interrupting long computation — computation proceed while signal handler writes

- long computation can't run while signal handler is
- signal handler is using its stack
- its registers are saved until signal handler returns
- ...

B: sigint_counter on stack — was global variable!

C: %rdi same before/after — yes

quiz Q3

A: append but not overwrite

- can set read/write/execute
- write allows overiwrte + append

B: letting programs run by the instructor read files created by certain students but not letting programs run by students read files created by certain other students

- probably shouldn't have used other in this option
- yes – can list user IDs, set read permissions

D: edit by partner

- yes – can list user IDs, set read+write

quiz Q4

A: one user read only ; others write

- not if that user is the owner, but yes if we make a group containing that user

B: one user read/write/change perms; others only read

- u=rw,og=r

C: one group read/one group write

- can't specify two groups

quiz Q5

0x1234 = 0b0001 0010 0011 0100

last 12 bits page offset

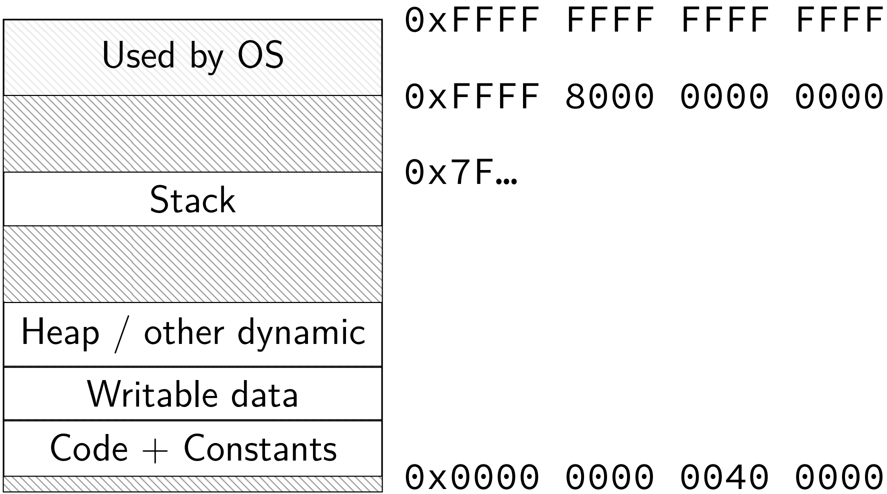
virtaul page number = 0001 = 1

is valid, physical page 2

combine with page offset:

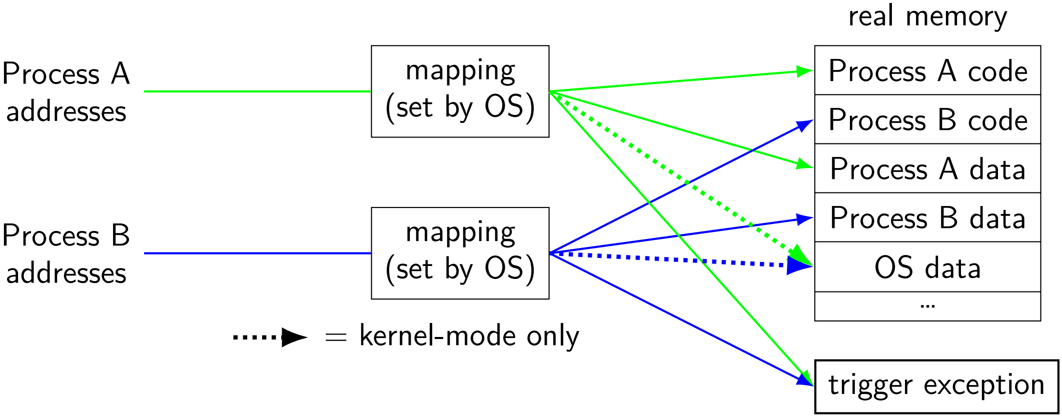
11 | 0010 0011 0100 = 0x2234

program memory



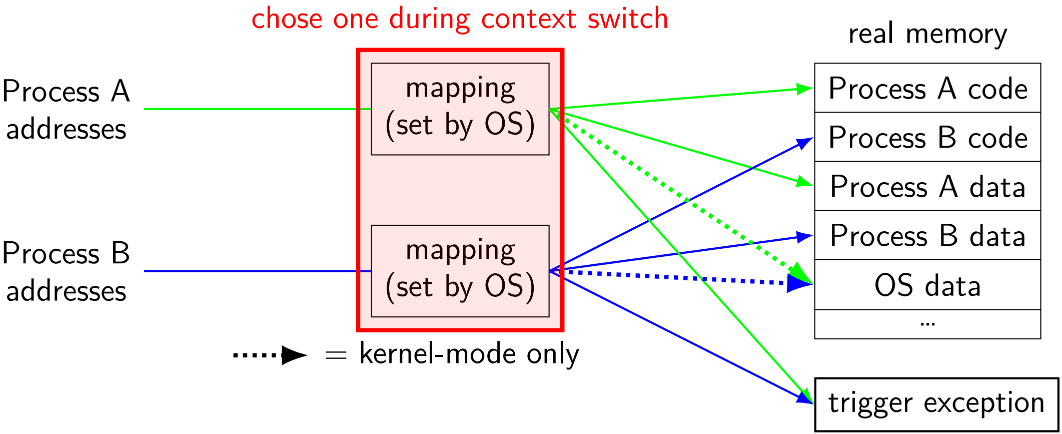
address spaces

illuision of dedicated memory

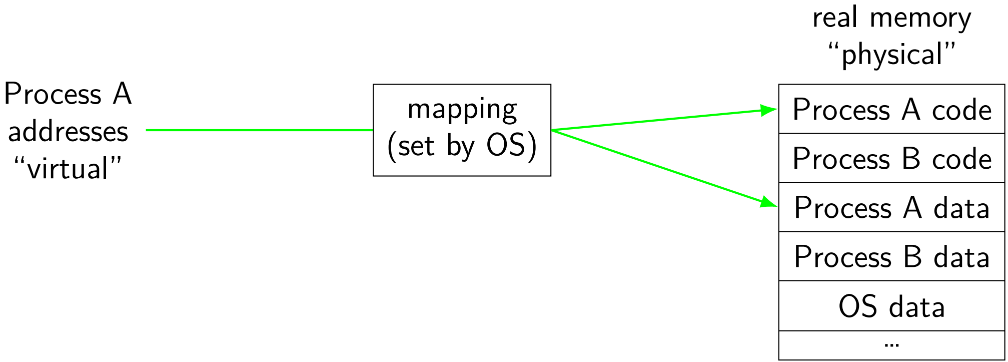


address spaces

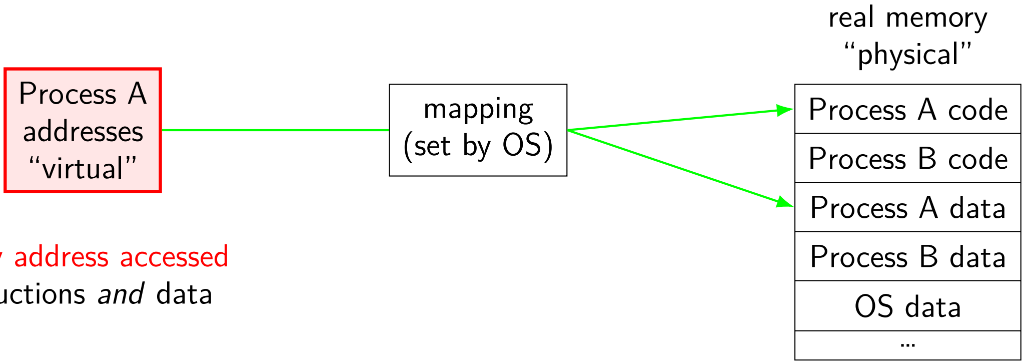
illuision of dedicated memory



address translation



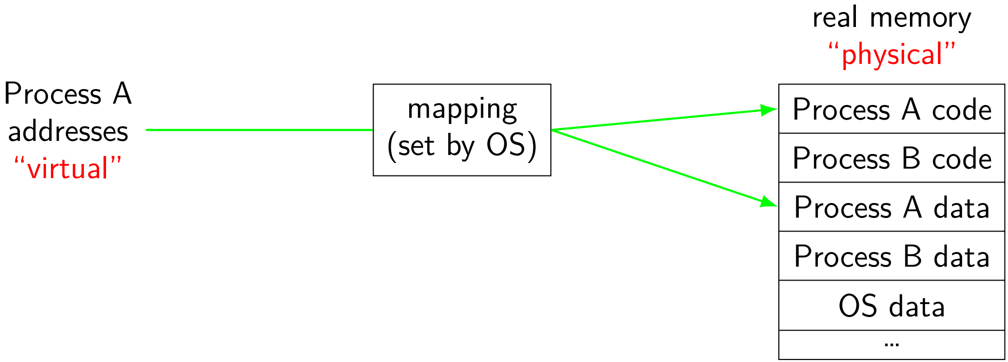
address translation



every address accessed

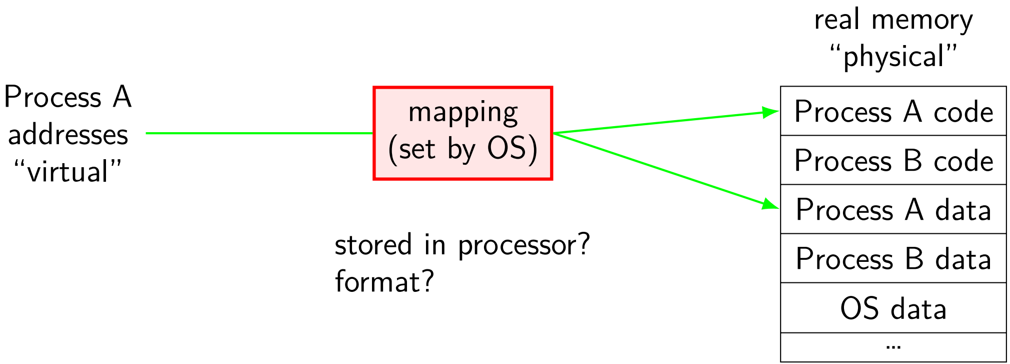
instructions *and* data

address translation



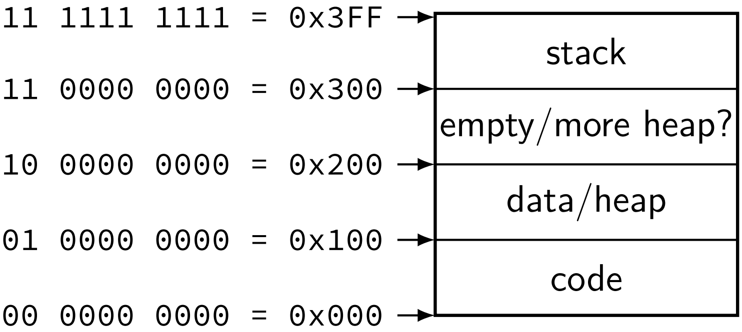
program addresses are ‘virtual’
real addresses are ‘physical’
can be different sizes!

address translation

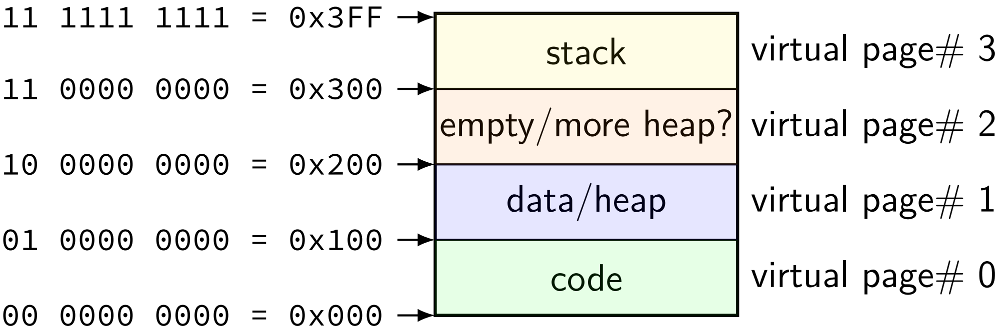


stored in processor?
format?

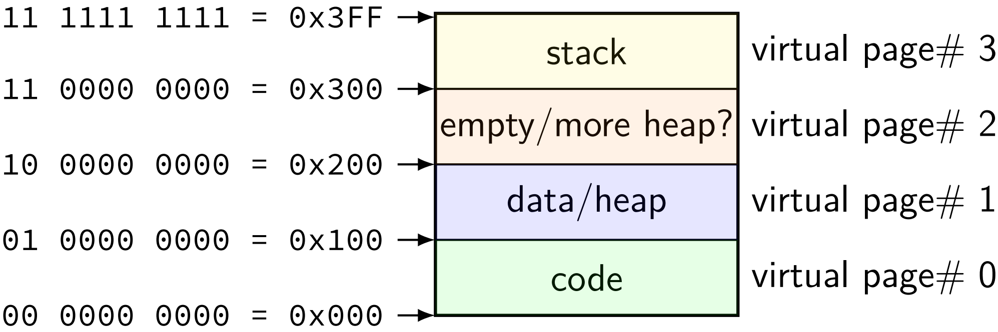
toy program memory



toy program memory

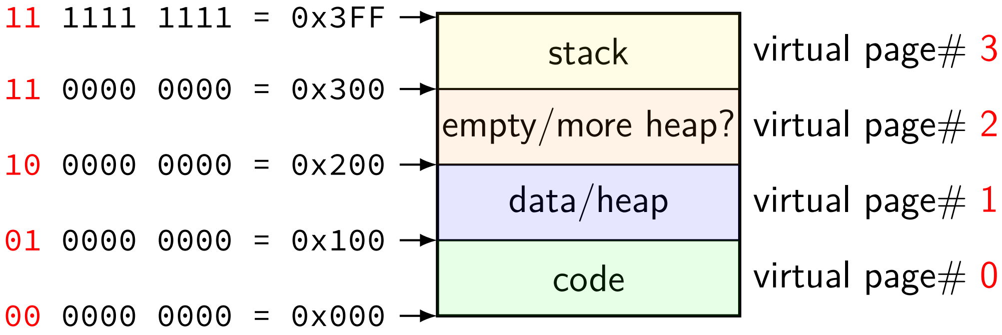


toy program memory



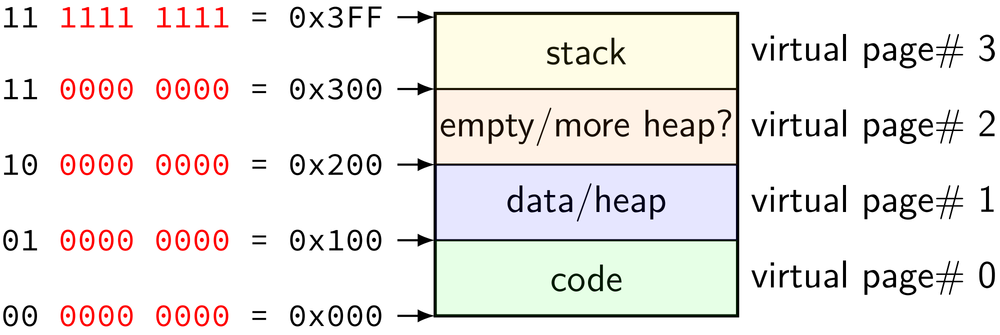
divide memory into pages (2^8 bytes in this case)
"virtual" = addresses the program sees

toy program memory



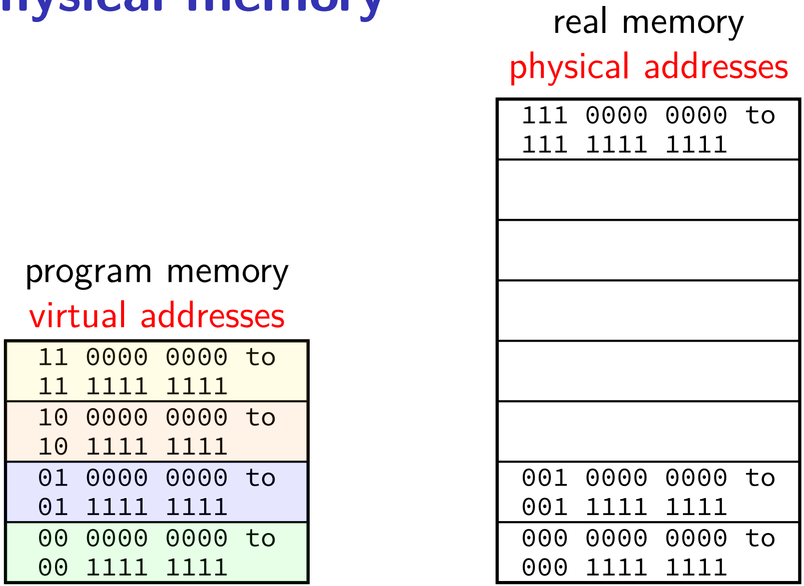
page number is upper bits of address
(because page size is power of two)

toy program memory



rest of address is called page offset

toy physical memory



toy physical memory

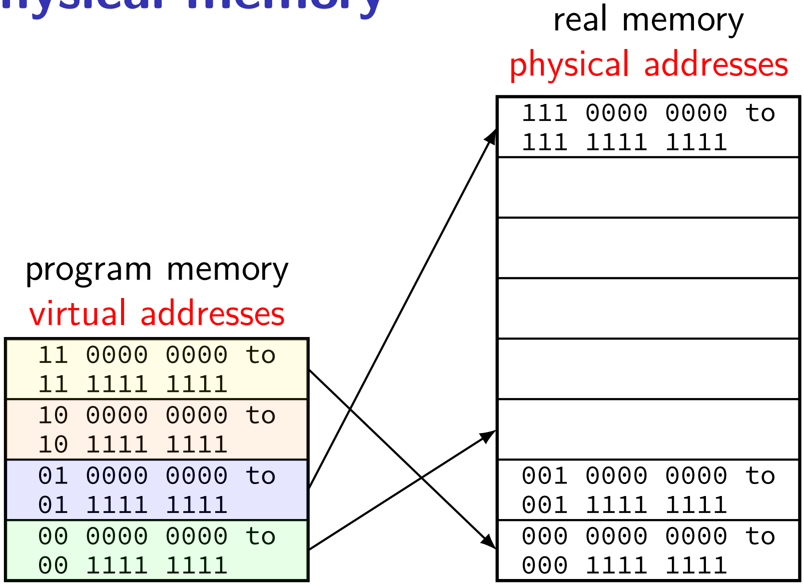
real memory
physical addresses

111 0000 0000 to 111 1111 1111	physical page 7
001 0000 0000 to 001 1111 1111	physical page 1
000 0000 0000 to 000 1111 1111	physical page 0

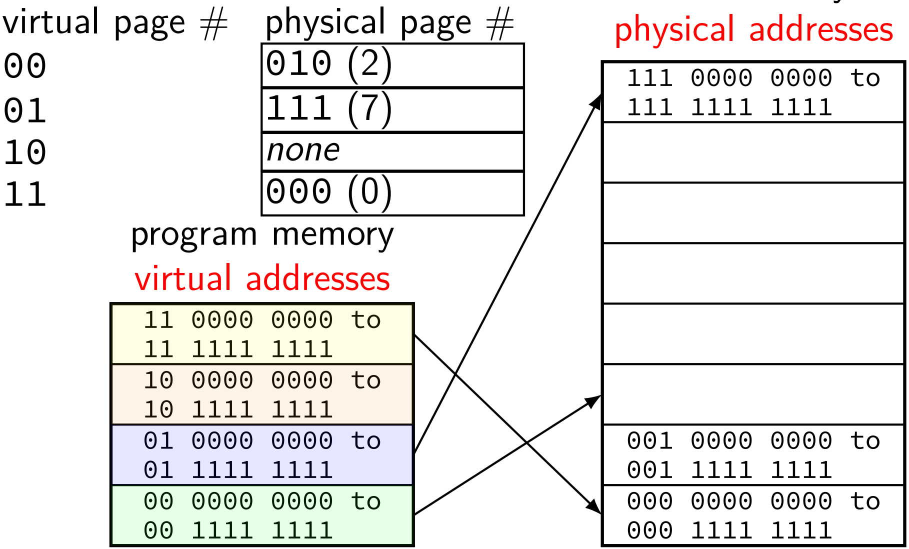
program memory
virtual addresses

11 0000 0000 to 11 1111 1111
10 0000 0000 to 10 1111 1111
01 0000 0000 to 01 1111 1111
00 0000 0000 to 00 1111 1111

toy physical memory



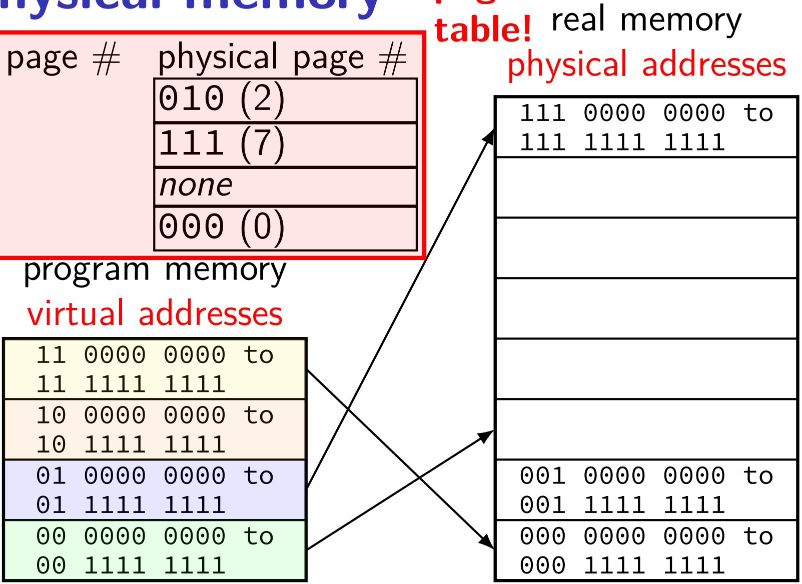
toy physical memory



toy physical memory

page table!

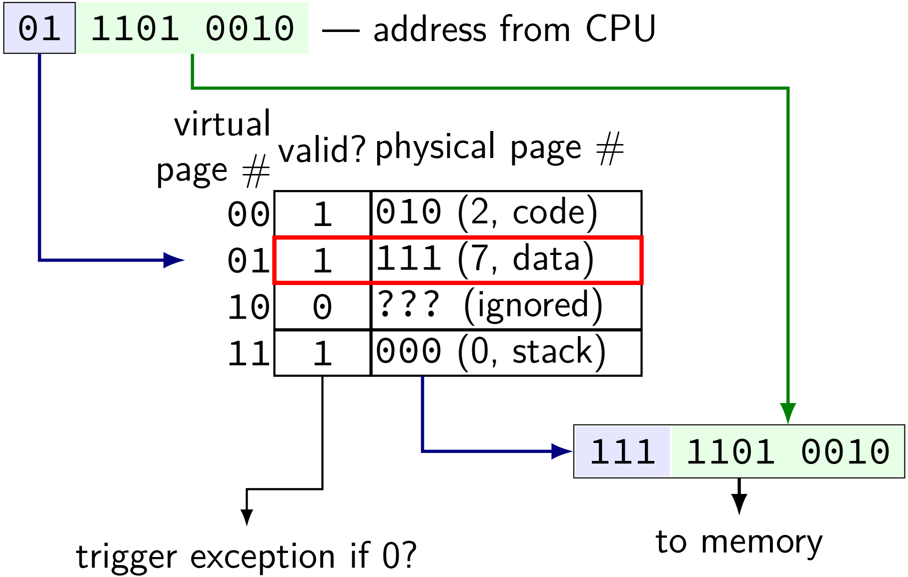
virtual page #	physical page #
00	010 (2)
01	111 (7)
10	*none*
11	000 (0)



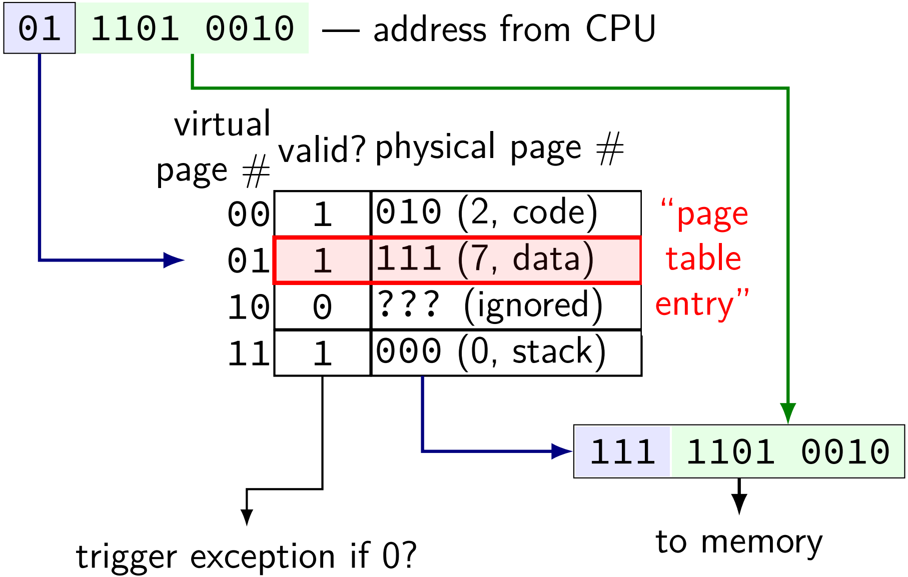
toy page table lookup

virtual page #	valid?	physical page #
00	1	010 (2, code)
01	1	111 (7, data)
10	0	??? (ignored)
11	1	000 (0, stack)

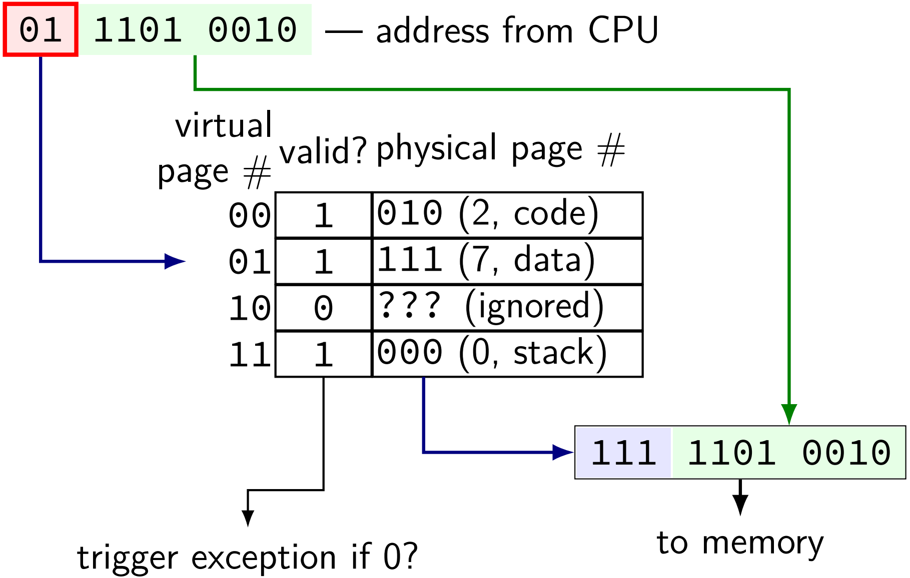
toy page table lookup



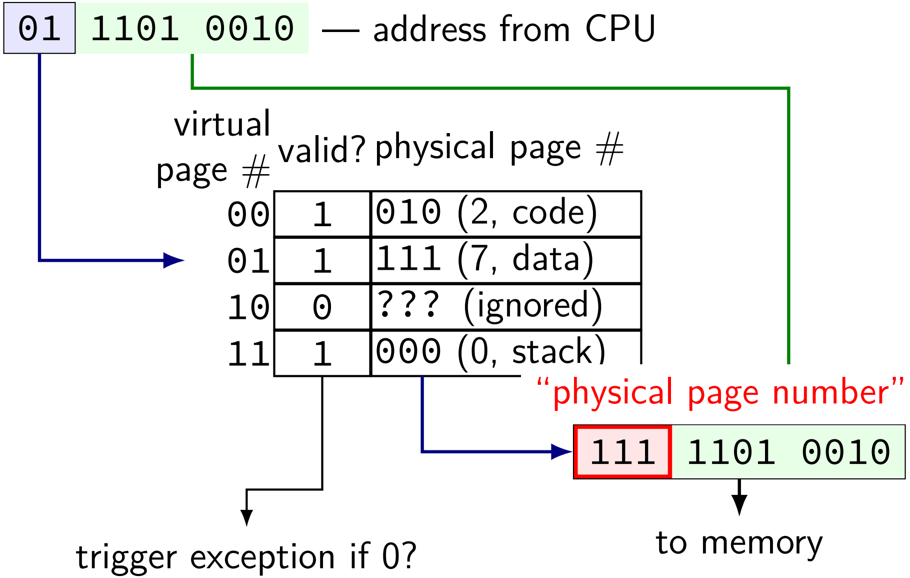
toy page table lookup



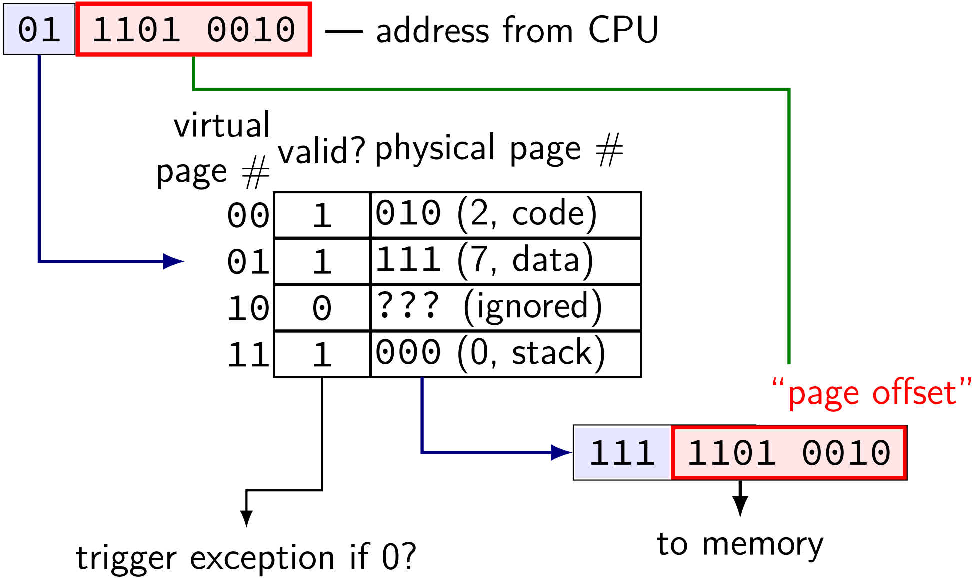
t “virtual page number” lookup



toy page table lookup



toy pag “page offset” ookup



switching page tables

part of context switch is changing the page table

extra privileged instructions

switching page tables

part of context switch is changing the page table

extra privileged instructions

where in memory is the code that does this switching?

switching page tables

part of context switch is changing the page table

extra privileged instructions

where in memory is the code that does this switching?

- probably have a page table entry pointing to it
- hopefully marked kernel-mode-only

switching page tables

part of context switch is changing the page table

extra privileged instructions

where in memory is the code that does this switching?

- probably have a page table entry pointing to it
- hopefully marked kernel-mode-only

code better not be modified by user program

- otherwise: uncontrolled way to "escape" user mode

on virtual address sizes

virtual address size = size of pointer?

often, but — sometimes part of pointer not used

example: typical x86-64 only use 48 bits

- rest of bits have fixed value

virtual address size is amount used for mapping

address space sizes

amount of stuff that can be addressed = address space size

based on number of unique addresses

e.g. 32-bit virtual address = 2^{32} byte virtual address space

e.g. 20-bit physical addresss = 2^{20} byte physical address space

address space sizes

amount of stuff that can be addressed = address space size

- based on number of unique addresses

e.g. 32-bit virtual address = 2^{32} byte virtual address space

e.g. 20-bit physical addresss = 2^{20} byte physical address space

what if my machine has 3GB of memory (not power of two)?

- not all addresses in physical address space are useful
- most common situation (since CPUs support having a lot of memory)

exercise: page counting

suppose 32-bit virtual (program) addresses

and each page is 4096 bytes (2^{12} bytes)

how many virtual pages?

exercise: page counting

suppose 32-bit virtual (program) addresses

and each page is 4096 bytes (2^{12} bytes)

how many virtual pages?

$2^{32}/2^{12} = 2^{20}$

exercise: page table size

suppose 32-bit virtual (program) addresses

suppose 30-bit physical (hardware) addresses

each page is 4096 bytes (2^{12} bytes)

pgae table entries have physical page #, valid bit, bit

how big is the page table (if laid out like ones we've seen)?

exercise: page table size

suppose 32-bit virtual (program) addresses

suppose 30-bit physical (hardware) addresses

each page is 4096 bytes (2^{12} bytes)

pgae table entries have physical page #, valid bit, bit

how big is the page table (if laid out like ones we've seen)?

2^{20} entries $\times (18 + 1)$ bits per entry

- issue: where can we store that?

exercise: address splitting

and each page is 4096 bytes (2^{12} bytes)

split the address `0x12345678` into page number and page offset:

exercise: address splitting

and each page is 4096 bytes (2^{12} bytes)

split the address `0x12345678` into page number and page offset:

page #: `0x12345`; offset: `0x678`

page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)

page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)

page table
base register

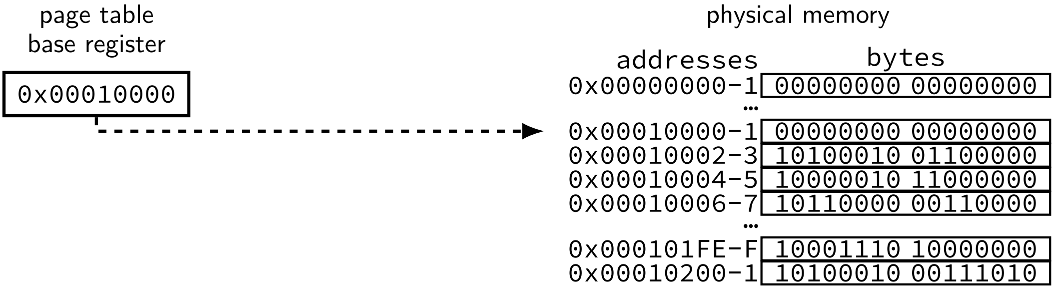
`0x00010000`

page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)



page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)

page table base register

`0x00010000`

physical memory

addresses	bytes
0x00000000-1	00000000 00000000
…	
0x00010000-1	00000000 00000000
0x00010002-3	10100010 01100000
0x00010004-5	10000010 11000000
0x00010006-7	10110000 00110000
…	
0x000101FE-F	10001110 10000000
0x00010200-1	10100010 00111010

page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)

page table base register

0x00010000

physical memory

addresses	bytes
0x00000000−1	00000000 00000000
…	
0x00010000−1	00000000 00000000
0x00010002−3	10100010 01100000
0x00010004−5	10000010 11000000
0x00010006−7	10110000 00110000
…	
0x000101FE−F	10001110 10000000
0x00010200−1	10100010 00111010

page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)

page table base register

`0x00010000`

physical memory

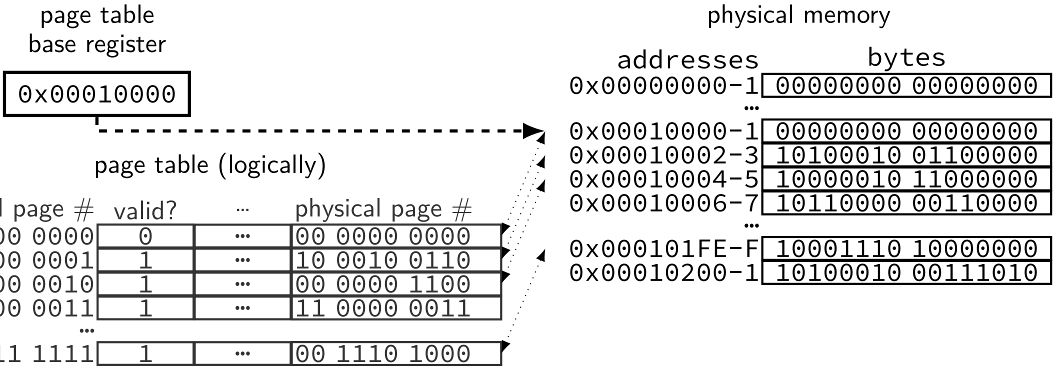
addresses	bytes
0x00000000-1	00000000 00000000
…	
0x00010000-1	00000000 00000000
0x00010002-3	10100010 01100000
0x00010004-5	10000010 11000000
0x00010006-7	10110000 00110000
…	
0x000101FE-F	10001110 10000000
0x00010200-1	10100010 00111010

page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)



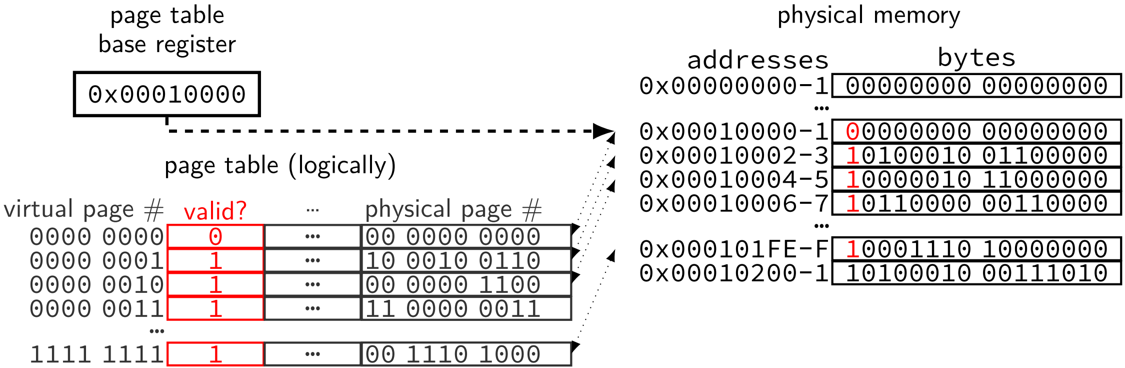
virtual page #	valid?	…	physical page #
0000 0000	0	…	00 0000 0000
0000 0001	1	…	10 0010 0110
0000 0010	1	…	00 0000 1100
0000 0011	1	…	11 0000 0011
…			
1111 1111	1	…	00 1110 1000

page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)

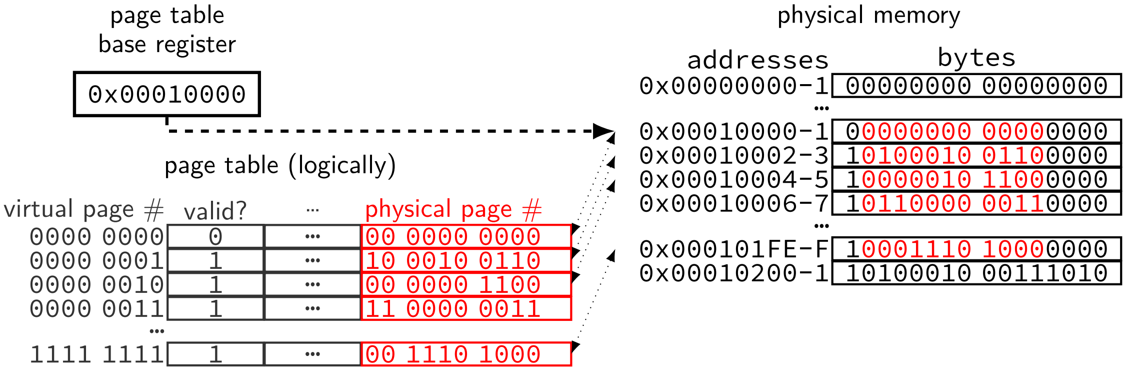


page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)



page tables in memory

where can processor store megabytes of page tables? in memory

page table entry layout (chosen by processor)

valid (bit 15)	physical page # (bits 4–14)	other bits and/or unused (bit 0-3)

page table base register

0x00010000

page table (logically)

virtual page #	valid?	...	physical page #
0000 0000	0	...	00 0000 0000
0000 0001	1	...	10 0010 0110
0000 0010	1	...	00 0000 1100
0000 0011	1	...	11 0000 0011
...			
1111 1111	1	...	00 1110 1000

physical memory

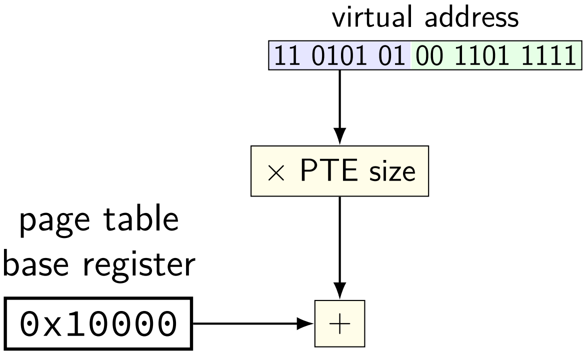
addresses	bytes
0x00000000-1	00000000 00000000
...	
0x00010000-1	00000000 00000000
0x00010002-3	10100010 01100000
0x00010004-5	10000010 11000000
0x00010006-7	10110000 00110000
...	
0x000101FE-F	10001110 10000000
0x00010200-1	10100010 00111010

memory access with page table

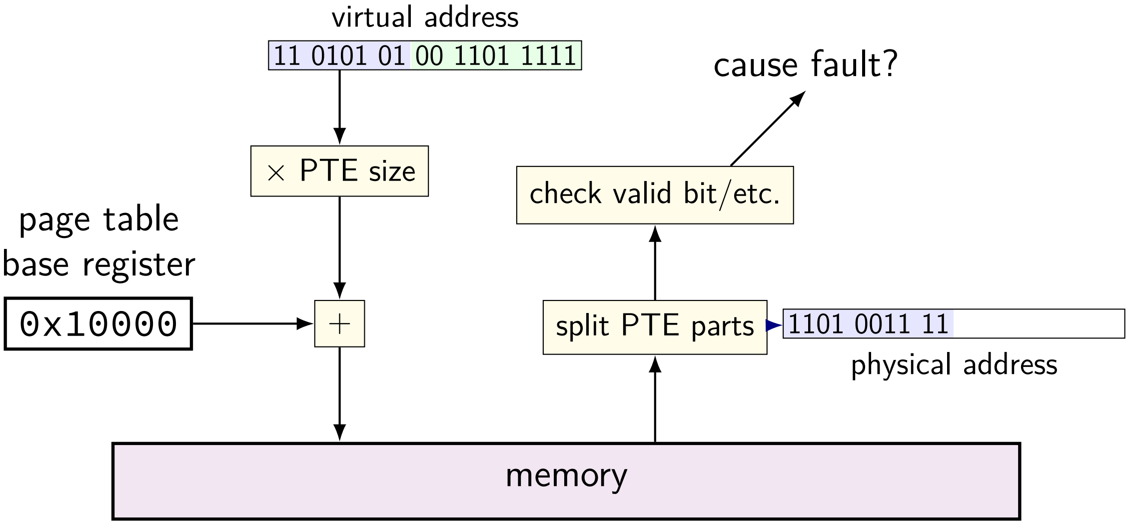
virtual address

11 0101 01	00 1101 1111

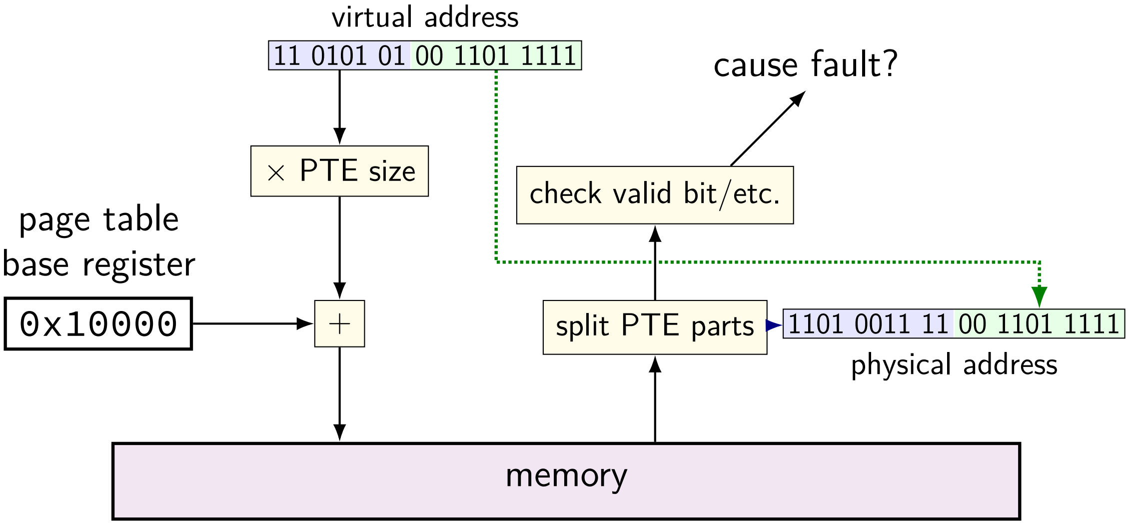
memory access with page table



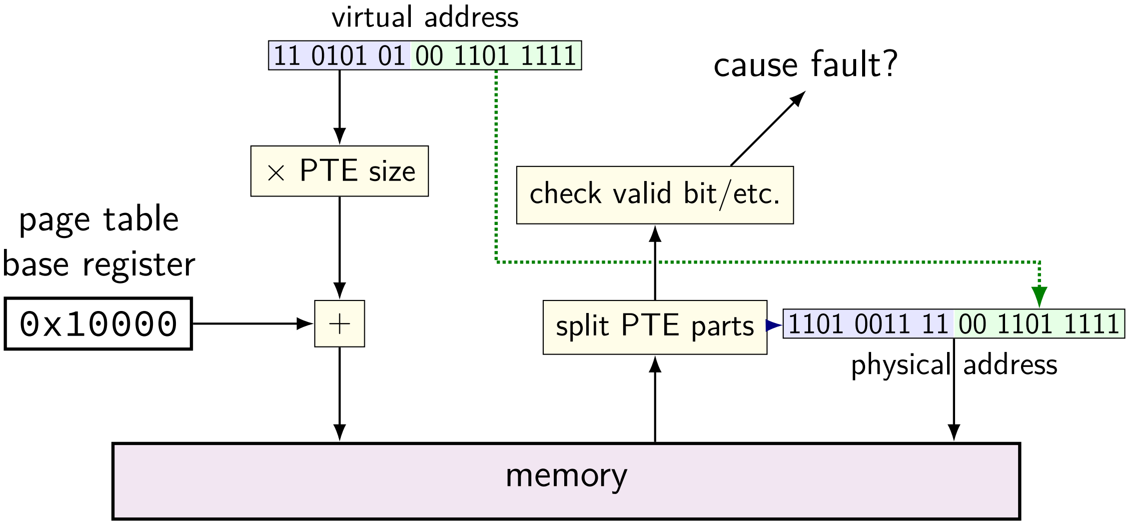
memory access with page table



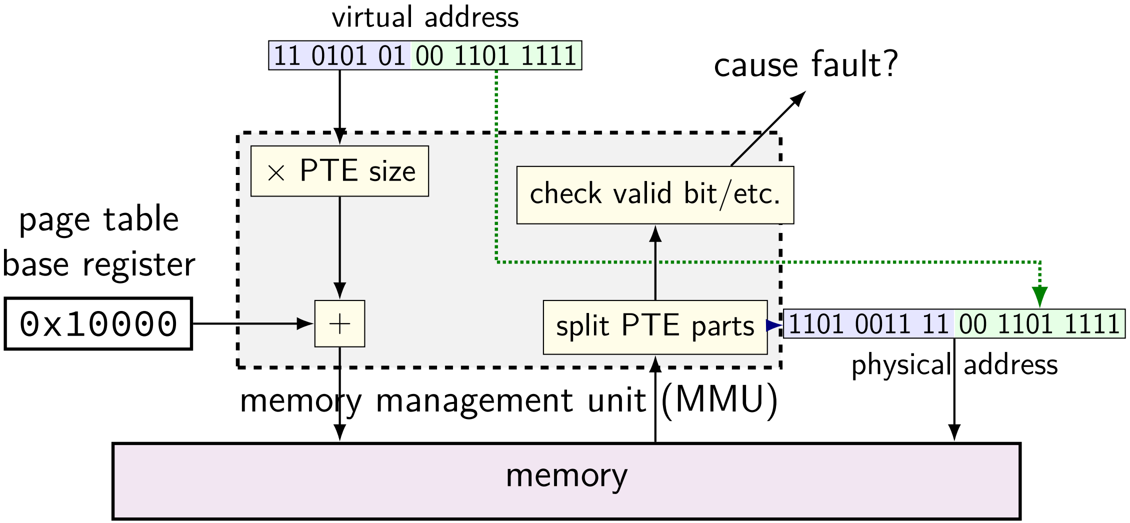
memory access with page table



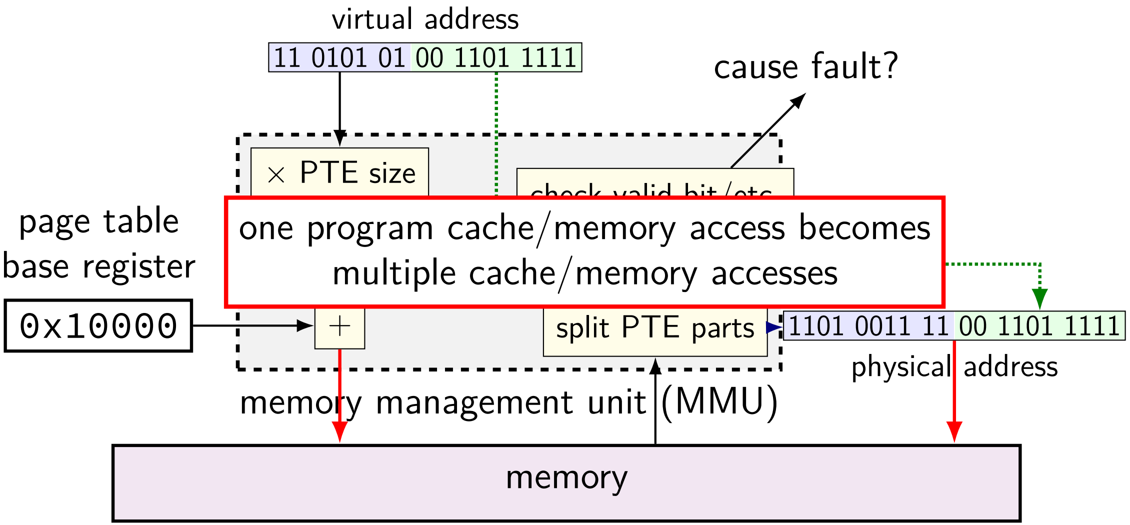
memory access with page table



memory access with page table

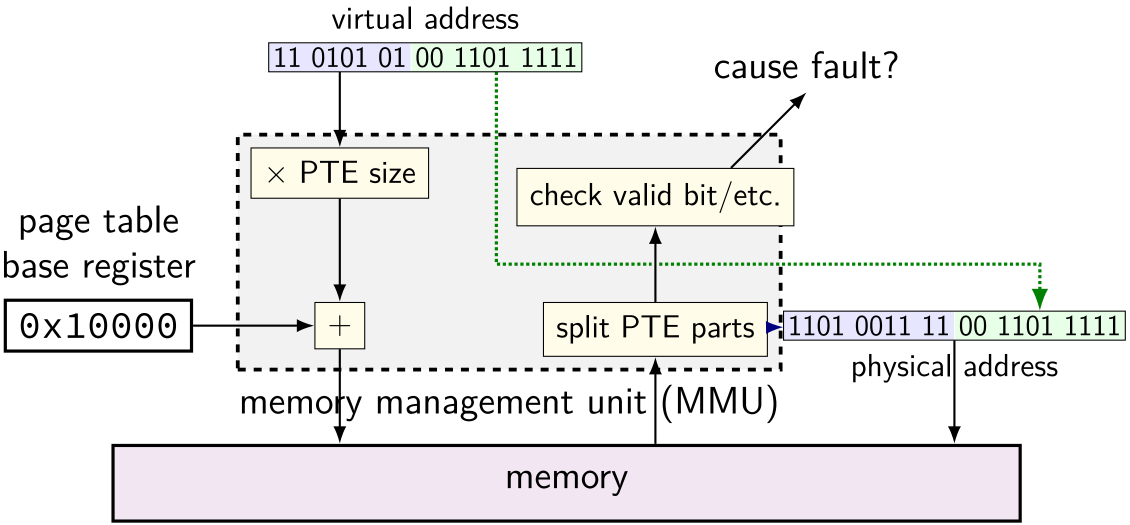


memory access with page table



one program cache/memory access becomes multiple cache/memory accesses

memory access with page table



exercise setup

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

page table

virtual page #	valid?	physical page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

exercise setup

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

page table

virtual page #	valid?	physical page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

phys. page 0

phys. page 1

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

(virtual addresses) `0x18` = ???; `0x03` = ???; `0x0A` = ???; `0x13` = ???

page table

virtual page #	valid?	physical page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

(virtual addresses) 0x18 = 00; 0x03 = ???; 0x0A = ???; 0x13 = ???

page table

virtual page #	valid?	physical page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

(virtual addresses) `0x18` = 00; `0x03` = 0x4A; `0x0A` = ???; `0x13` = ???

page table

virtual page #	valid?	physical page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

(virtual addresses) `0x18` = `00`; `0x03` = 0x4A; `0x0A` = 0xDC; `0x13` = ???

page table

virtual page #	valid?	physical page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

exercise

5-bit virtual addresses, 6-bit physical addresses, 8-byte pages

(virtual addresses) `0x18` = `00`; `0x03` = 0x4A; `0x0A` = `0xDC`; `0x13` = fault

page table

virtual page #	valid?	physical page #
00	1	010
01	1	111
10	0	000
11	1	000

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

1-level exercise (1)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

page table base register `0x20`; translate virtual address `0x31`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

1-level exercise (1)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

page table base register `0x20`; translate virtual address `0x31`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

0x31 = 11 0001

PTE addr:

0x20 + 6 ×1 = 0x26

PTE value:

0xF6 = 1111 0110

PPN 111, valid 1

M[111 001] = M[0x39]

→ 0x0C

1-level exercise (1)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

page table base register `0x20`; translate virtual address `0x31`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x31` = `11 0001`

PTE addr:

`0x20` + 6 ×1 = `0x26`

PTE value:

`0xF6` = `1111 0110`

PPN `111`, valid 1

M[`111 001`] = M[`0x39`]

→ `0x0C`

1-level exercise (1)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other;

page table base register `0x20`; translate virtual address `0x31`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x31` = `11 0001`

PTE addr:

`0x20` + 6 ×1 = `0x26`

PTE value:

`0xF6` = `1111 0110`

PPN `111`, valid `1`

M[`111 001`] = M[`0x39`]

→ `0x0C`

1-level exercise (2)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other

page table base register `0x20`; translate virtual address `0x12`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

1-level exercise (2)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other

page table base register `0x20`; translate virtual address `0x12`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x12` = `01 0010`

PTE addr:

`0x20` + 2 × 1 = `0x22`

PTE value:

`0xD2` = `1101 0010`

PPN `110`, valid `1`

M[`110 010`] = **M[**`0x32`]

→ `0xBA`

1-level exercise (2)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other

page table base register `0x20`; translate virtual address `0x12`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x12` = `01 0010`

PTE addr:

`0x20` + 2 × 1 = `0x22`

PTE value:

`0xD2` = `1101 0010`

PPN `110`, valid 1

M[`110 010`] = **M[**`0x32`**]**

→ `0xBA`

1-level exercise (2)

6-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 other

page table base register `0x20`; translate virtual address `0x12`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	F4 F5 F6 F7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	CB 0B CB 0B
0x38-B	DC 0C DC 0C
0x3C-F	EC 0C EC 0C

`0x12` = `01 0010`

PTE addr:

`0x20` + 2 ×1 = `0x22`

PTE value:

`0xD2` = `1101 0010`

PPN `110`, valid `1`

M[`110 010`] = **M[**`0x32`]

→ `0xBA`

exercise: 64-bit system

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

exercise: 64-bit system

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

top 16 bits of 64-bit addresses not used for translation

exercise: 64-bit system

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

exercise: how many page table entries? (assuming page table like shown before)

exercise: how large are physical page numbers?

exercise: 64-bit system

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

exercise: how many page table entries? (assuming page table like shown before) $2^{48}/2^{12} = 2^{36}$ entries

exercise: how large are physical page numbers? $39 - 12 = 27$ bits

exercise: 64-bit system

my desktop: 39-bit physical addresses; 48-bit virtual addresses

4096 byte pages

exercise: how many page table entries? (assuming page table like shown before) $2^{48}/2^{12} = 2^{36}$ entries

exercise: how large are physical page numbers? $39 - 12 = 27$ bits

page table entries are 8 bytes (room for expansion, metadata)

- trick: power of two size makes table lookup faster

would take up 2^{39} bytes?? (512GB??)

backup slides

some notes on timing HW (1)

timings.txt — file for us to read

if you have lots of data files, can submit separately now

originally wanted ‘time a function’, ‘time a syscall’

choose `getpid` as syscall

turns out *sometimes* (not on my system) getpid only makes syscall the first time

- remembers pid the other times

some notes on timing HW (2)

yes, getting consistent timings is tricky