introduction / layers of abstraction

lecture logistics

lectures via Zoom

there will be a recording

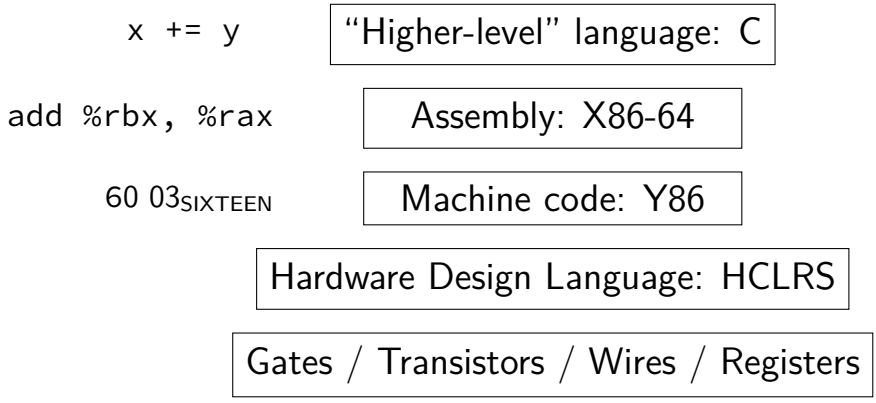
I will watch the chat

probably the best way to ask questions

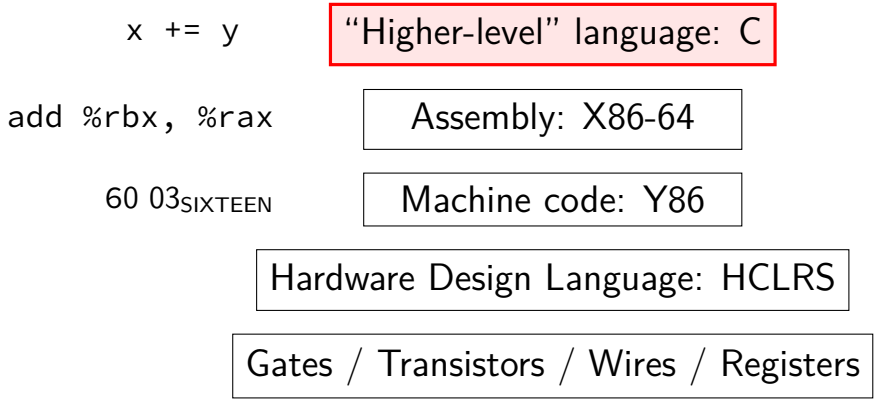
also know if you click “raise hand”

introduction / layers of abstraction

layers of abstraction



layers of abstraction



why C?

almost a subset of C++

- notably removes classes, new/delete, iostreams
- other changes, too, so C code often not valid C++ code

direct correspondence to assembly

why C?

almost a subset of C++

notably removes classes, new/delete, iostreams
other changes, too, so C code often not valid C++ code

direct correspondence to assembly

Should help you understand machine!
Manual translation to assembly

why C?

almost a subset of C++

- notably removes classes, new/delete, iostreams
- other changes, too, so C code often not valid C++ code

direct correspondence to assembly

> But "clever" (optimizing) compiler might be confusingly indirect instead

homework: C environment

get Unix-like environment with a C compiler

will have department accounts, hopefully by end of week

- portal.cs.virginia.edu or NX
- instructions off course website (Collab)

some other options:

- Linux (native or VM)
  - 2150 VM image should work
- some assignments can use OS X natively
- some assignments can Windows Subsystem for Linux natively

assignment compatibility

supported platform: department machines

many use laptops

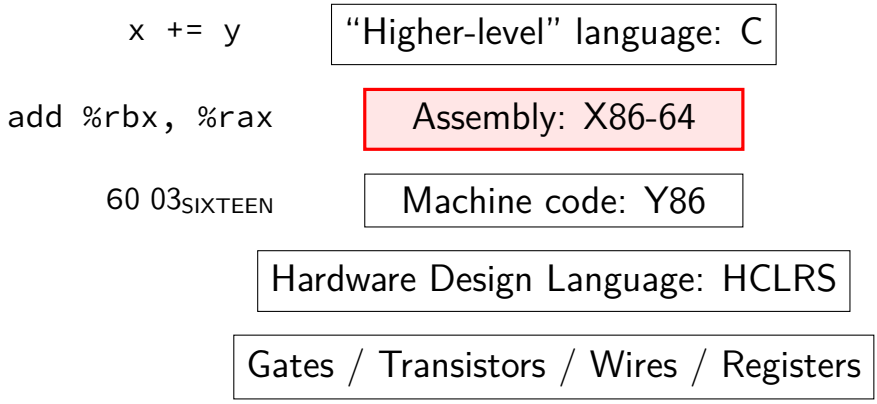
trouble? we'll say to use department machines

most assignments: C and Unix-like environment

also: tool written in Rust — but we'll provide binaries

previously written in D + needed D compiler

layers of abstraction

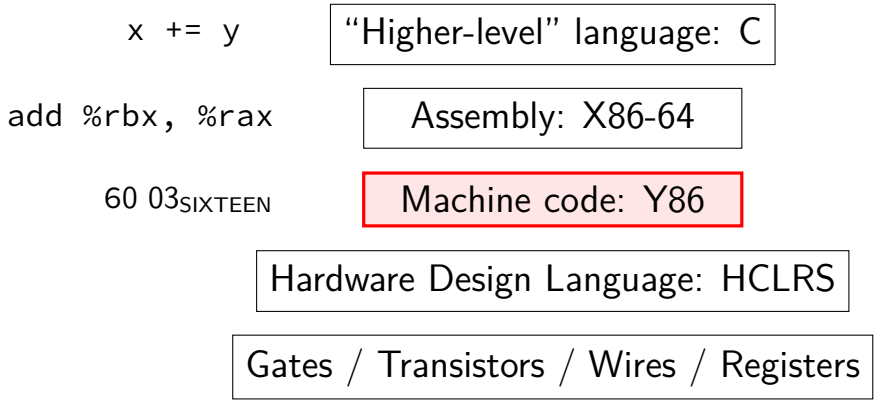


X86-64 assembly

in theory, you know this (CS 2150)

in reality, ...

layers of abstraction



Y86-64??

Y86: our textbook's X86-64 subset

- hope: leverage 2150 assembly knowledge

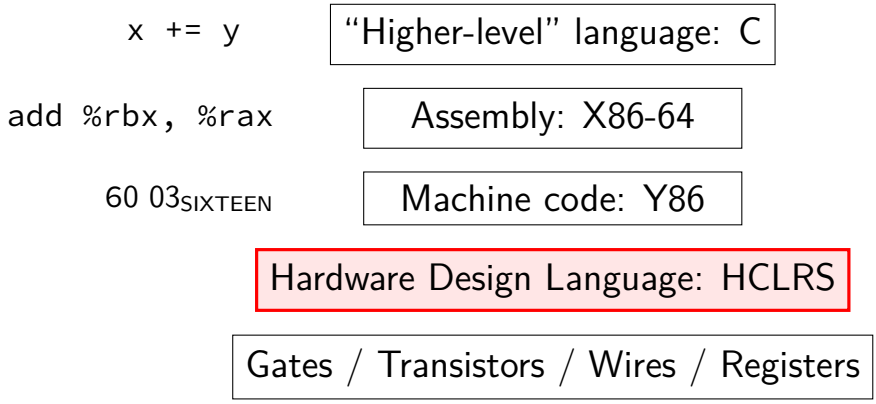
much simpler than real X86-64 encoding

- (which we will not cover)

not as simple as 2150's IBCM

- variable-length encoding
- more than one register
- full conditional jumps
- stack-manipulation instructions

layers of abstraction



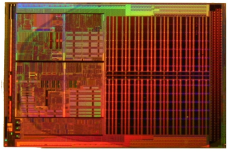
textbook

Computer Systems: A Programmer's Perspective

recommended — HCL assignments follow pretty closely

(useful, but less important for other topics)

processors and memory

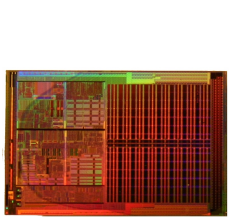


processor

memory

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



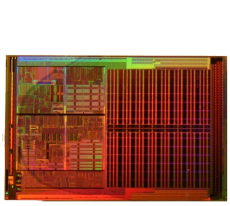
processor

bus
send address + send or get data

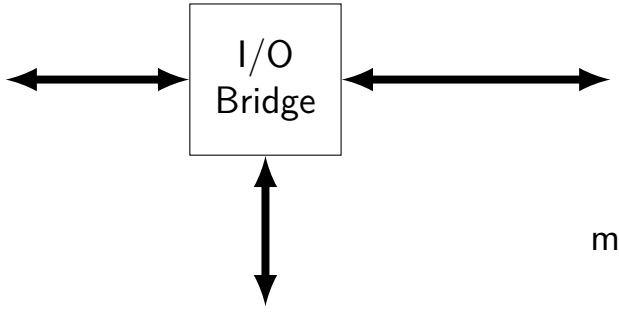
memory

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



processor



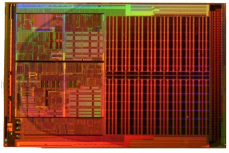
memory

to I/O devices

keyboard, mouse, wifi, ...

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



processor

bus
send address + send or get data
(machine code/text/number...)

Bridge

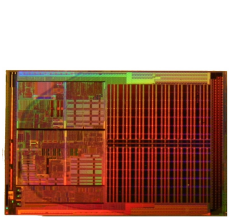
memory

to I/O devices

keyboard, mouse, wifi, ...

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



processor

CPU: send PC: `0x04000`

MEM: send machine code: `pushq %rbp`

Bridge

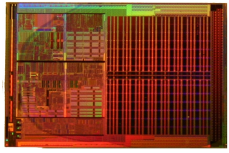
memory

to I/O devices

keyboard, mouse, wifi, ...

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



processor

CPU: send PC: `0x04000`

MEM: send machine code: `pushq %rbp`

CPU: send `0x7fff830` (stack), `0x1234` (rbp)

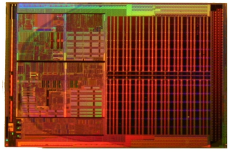
memory

to I/O devices

keyboard, mouse, wifi, ...

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



processor

CPU: send PC: `0x04000`

MEM: send machine code: `pushq %rbp`

CPU: send `0x7fff830` (stack), `0x1234` (rbp)

MEM: stored it

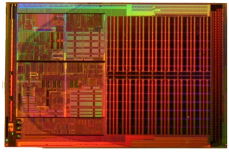
memory

to I/O devices

keyboard, mouse, wifi, ...

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



processor

CPU: send PC: `0x04000`

MEM: send machine code: `pushq %rbp`

CPU: send `0x7fff830` (stack), `0x1234` (rbp)

MEM: stored it

memory

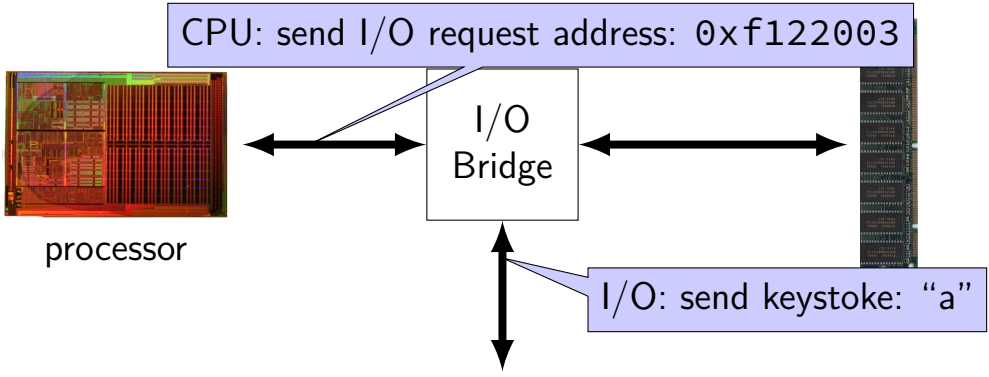
CPU: next PC: `0x04001`

to I/O devices

keyboard, mouse, wifi, ...

Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

processors and memory



Images:
Single core Opteron 8xx die: Dg2fer at the German language Wikipedia, via Wikimedia Commons
SDRAM by Arnaud 25, via Wikimedia Commons

goals/other topics

understand how hardware works for...

program performance

what compilers are/do

weird program behaviors (segfaults, etc.)

goals/other topics

understand how hardware works for...

program performance

what compilers are/do

weird program behaviors (segfaults, etc.)

program performance

naive model:

one instruction = one time unit

number of instructions matters, but ...

program performance: issues

parallelism

- fast hardware is parallel
- needs multiple things to do

caching

- accessing things recently accessed is faster
- need reuse of data/code

(more in other classes: algorithmic efficiency)

goals/other topics

understand how hardware works for...

program performance

what compilers are/do

weird program behaviors (segfaults, etc.)

what compilers are/do

understanding weird compiler/linker rrors

if you want to make compilers

debugging applications

goals/other topics

understand how hardware works for...

program performance

what compilers are/do

weird program behaviors (segfaults, etc.)

weird program behaviors

what is a segmentation fault really?

how does the operating system interact with programs?

if you want to handle them — writing OSs

coursework

labs — grading: did you make reasonable progress?

- collaboration permitted

homework assignments — introduced by lab (mostly)

- due at 9:30am lab day
- complete individually

weekly quizzes

final exam

coursework

labs — grading: did you make reasonable progress?

- collaboration permitted

homework assignments — introduced by lab (mostly)

- due at 9:30am lab day
- complete individually

weekly quizzes

final exam

textbook

Computer Systems: A Programmer's Perspective

recommended — HCL assignments follow pretty closely

(useful, but less important for other topics)

on lecture/lab/HW synchronization

labs/HWs not quite synchronized with lectures

main problem: want to cover material **before you need it** in lab/HW

quizzes?

linked off course website (demo Thursday)

released Thursday evening, due Tuesday before *first* lecture

from lecture that week

sometimes also next week's readings

(for parts of course where we follow textbook closely)

two lowest quizzes dropped

late policy

exceptional circumstance? contact us.

otherwise, for homeworks only:

- -10% 0 to 48 hours late
- -15% 48 to 72 hours late
- -100% otherwise

late quizzes, labs: no

- we release answers
- talk to us if illness, etc.

getting help tools

lab + OH: Discord (voice+text chat)

non-real-time help: Piazza (discussion forum)

on Discord

instructions on website

you could have a separate account from other uses of Discord

lab/office hours logistics (1)

labs+OH: held on Discord

public channels for you to chat (voice + text)

queue for TA help (DEMO)

- shared between OH/lab

lab/office hours logistics (2)

TA help primarily via voice channels

private and public channels

- indicate on queue if help needs to be public
- also indicate if you can't do voice

also channels for student-led text+voice discussion

- TAs *might* chime in
- primary use: students helping each other
- especially: find someone to talk to lab about

on the office hour queue

except for first three slots, queue is sorted by last time helped

we may reset those first three slots between office hours

goal 1: being on the queue overnight won't help you

goal 2: try to spread out the TA help

office hour calendar

office hours will be posted on calendar on the website

your TODO list

Discord account working

department account and/or C environment working

- department accounts should happen by this weekend

before lab next week

grading

Quizzes: 30%

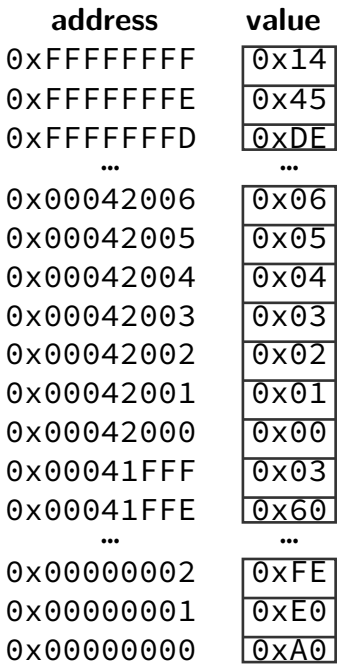
Homeworks: 40%

Labs: 15%

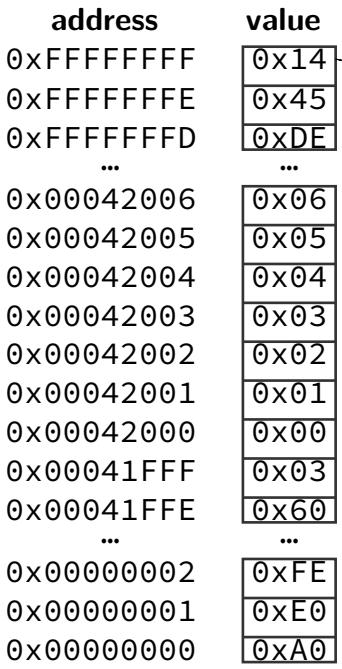
Final Exam: 15%

quiz demo

memory


address
value
0xFFFFFFFF
0x14
0xFFFFFFFE
0x45
0xFFFFFFFD
0xDE
…
…
0x00042006
0x06
0x00042005
0x05
0x00042004
0x04
0x00042003
0x03
0x00042002
0x02
0x00042001
0x01
0x00042000
0x00
0x00041FFF
0x03
0x00041FFE
0x60
…
…
0x00000002
0xFE
0x00000001
0xE0
0x00000000
0xA0

memory



array of bytes (byte = 8 bits)
CPU interprets based on how accessed

memory

address	value
0xFFFFFFFF	0x14
0xFFFFFFFE	0x45
0xFFFFFFFD	0xDE
…	…
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
…	…
0x00000002	0xFE
0x00000001	0xE0
0x00000000	0xA0

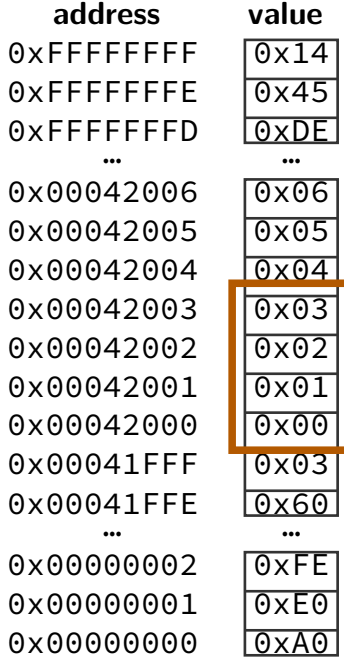
address	value
0x00000000	0xA0
0x00000001	0xE0
0x00000002	0xFE
…	…
0x00041FFE	0x60
0x00041FFF	0x03
0x00042000	0x00
0x00042001	0x01
0x00042002	0x02
0x00042003	0x03
0x00042004	0x04
0x00042005	0x05
0x00042006	0x06
…	…
0xFFFFFFFD	0xDE
0xFFFFFFFE	0x45
0xFFFFFFFF	0x14

endianness

address	value
0xFFFFFFFF	0x14
0xFFFFFFFE	0x45
0xFFFFFFFD	0xDE
…	…
0x00042006	0x06
0x00042005	0x05
0x00042004	0x04
0x00042003	0x03
0x00042002	0x02
0x00042001	0x01
0x00042000	0x00
0x00041FFF	0x03
0x00041FFE	0x60
…	…
0x00000002	0xFE
0x00000001	0xE0
0x00000000	0xA0

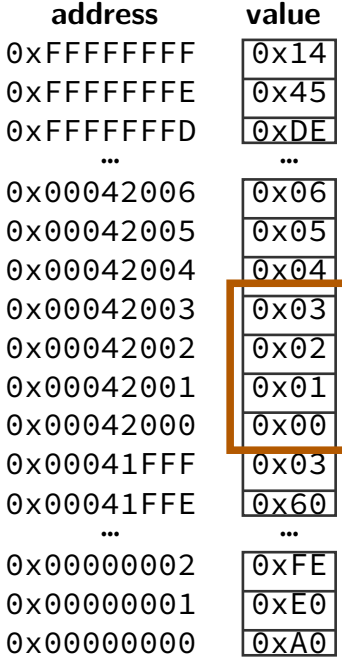
```
int *x = (int*)0x42000;
printf("%d\n", *x);
```

endianness




```
int *x = (int*)0x42000;
printf("%d\n", *x);
```

endianness

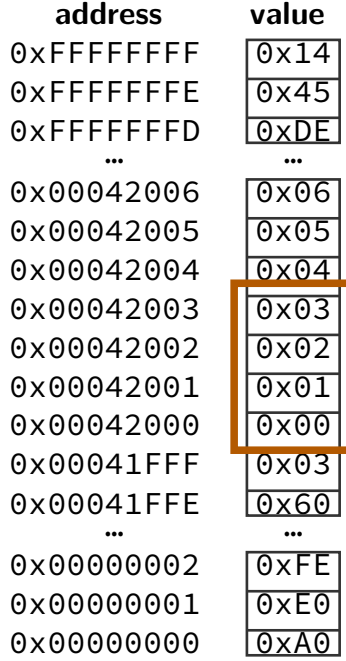



```
int *x = (int*)0x42000;
printf("%d\n", *x);
```

0x03020100 = 50462976

0x00010203 = 66051

endianness




```
int *x = (int*)0x42000;
printf("%d\n", *x);
```

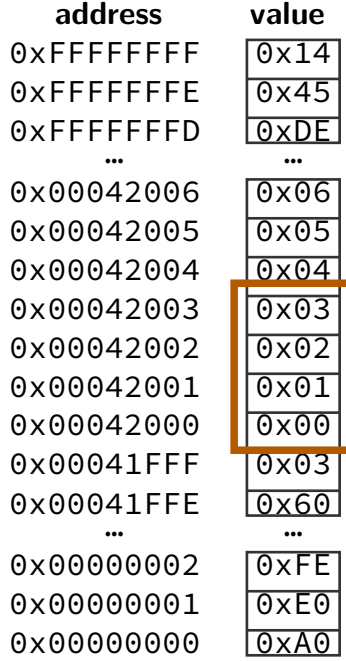
0x03020100 = 50462976

little endian
(least significant byte has lowest address)

0x00010203 = 66051

big endian
(most significant byte has lowest address)

endianness




```
int *x = (int*)0x42000;
printf("%d\n", *x);
```

0x03020100 = 50462976

little endian
(least significant byte has lowest address)

0x00010203 = 66051

big endian
(most significant byte has lowest address)

exericse

buffer

```
unsigned char buffer[8] =
    { 0, 0, /* ..., */ 0 };
/* uint32_t = 32-bit unsigned int */
uint32_t value1 = 0x12345678;
uint32_t value2 = 0x9ABCDEF0;
unsigned char *ptr_value1 = (unsigned char *) &value1;
unsigned char *ptr_value2 = (unsigned char *) &value2;
for (int i = 0; i < 4; ++i) { /* copy value1/2 into buffer */
    buffer[i] = ptr_value1[i];
    buffer[i+4] = ptr_value2[i];
}
for (int i = 0; i < 4; ++i) { /* copy buffer[1..5] into value1 */
    ptr_value1[i] = buffer[i+1];
}
```

What is `value1` after this runs on a little-endian system?

A. `0x0F654321` **B.** `0x123456F0` **C.** `0x3456789A`
D. `0x345678F0` **E.** `0x9A123456` **F.** `0x9A785634`
G. `0xF0123456` **H.** `0xF2345678` **I.** something else

exericse

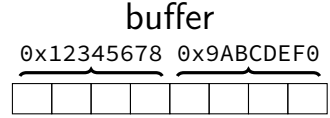
buffer

```
unsigned char buffer[8] =
    { 0, 0, /* ..., */ 0 };
/* uint32_t = 32-bit unsigned int */
uint32_t value1 = 0x12345678;
uint32_t value2 = 0x9ABCDEF0;
unsigned char *ptr_value1 = (unsigned char *) &value1;
unsigned char *ptr_value2 = (unsigned char *) &value2;
for (int i = 0; i < 4; ++i) { /* copy value1/2 into buffer */
    buffer[i] = ptr_value1[i];
    buffer[i+4] = ptr_value2[i];
}
for (int i = 0; i < 4; ++i) { /* copy buffer[1..5] into value1 */
    ptr_value1[i] = buffer[i+1];
}
```

What is `value1` after this runs on a little-endian system?

A. `0x0F654321` **B.** `0x123456F0` **C.** `0x3456789A`
D. `0x345678F0` **E.** `0x9A123456` **F.** `0x9A785634`
G. `0xF0123456` **H.** `0xF2345678` **I.** something else

exericse

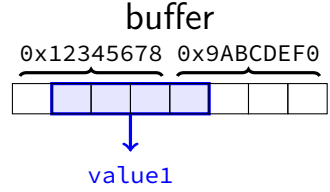



```
unsigned char buffer[8] =
    { 0, 0, /* ..., */ 0 };
/* uint32_t = 32-bit unsigned int */
uint32_t value1 = 0x12345678;
uint32_t value2 = 0x9ABCDEF0;
unsigned char *ptr_value1 = (unsigned char *) &value1;
unsigned char *ptr_value2 = (unsigned char *) &value2;
for (int i = 0; i < 4; ++i) { /* copy value1/2 into buffer */
    buffer[i] = ptr_value1[i];
    buffer[i+4] = ptr_value2[i];
}
for (int i = 0; i < 4; ++i) { /* copy buffer[1..5] into value1 */
    ptr_value1[i] = buffer[i+1];
}
```

What is `value1` after this runs on a little-endian system?

A. `0x0F654321` **B.** `0x123456F0` **C.** `0x3456789A`
D. `0x345678F0` **E.** `0x9A123456` **F.** `0x9A785634`
G. `0xF0123456` **H.** `0xF2345678` **I.** something else

exericse

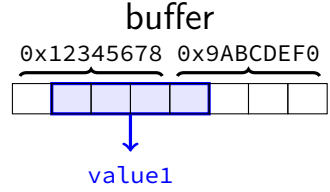



```
unsigned char buffer[8] =
    { 0, 0, /* ..., */ 0 };
/* uint32_t = 32-bit unsigned int */
uint32_t value1 = 0x12345678;
uint32_t value2 = 0x9ABCDEF0;
unsigned char *ptr_value1 = (unsigned char *) &value1;
unsigned char *ptr_value2 = (unsigned char *) &value2;
for (int i = 0; i < 4; ++i) { /* copy value1/2 into buffer */
    buffer[i] = ptr_value1[i];
    buffer[i+4] = ptr_value2[i];
}
for (int i = 0; i < 4; ++i) { /* copy buffer[1..5] into value1 */
    ptr_value1[i] = buffer[i+1];
}
```

What is `value1` after this runs on a little-endian system?

A. `0x0F654321` **B.** `0x123456F0` **C.** `0x3456789A`
D. `0x345678F0` **E.** `0x9A123456` **F.** `0x9A785634`
G. `0xF0123456` **H.** `0xF2345678` **I.** something else

exericse




```
unsigned char buffer[8] =
    { 0, 0, /* ..., */ 0 };
/* uint32_t = 32-bit unsigned int */
uint32_t value1 = 0x12345678;
uint32_t value2 = 0x9ABCDEF0;
unsigned char *ptr_value1 = (unsigned char *) &value1;
unsigned char *ptr_value2 = (unsigned char *) &value2;
for (int i = 0; i < 4; ++i) { /* copy value1/2 into buffer */
    buffer[i] = ptr_value1[i];
    buffer[i+4] = ptr_value2[i];
}
for (int i = 0; i < 4; ++i) { /* copy buffer[1..5] into value1 */
    ptr_value1[i] = buffer[i+1];
}
```

What is `value1` after this runs on a little-endian system?

A. `0x0F654321` **B.** `0x123456F0` **C.** `0x3456789A`
D. `0x345678F0` **E.** `0x9A123456` **F.** `0x9A785634`
G. `0xF0123456` **H.** `0xF2345678` **I.** something else

backup slides