clicker.cs.illinois.edu

Q1

~Code~
340

CS 340

Central Processing Unit

Updates

1. MP 3 - PNG, due Tuesday after exam
    a. Recommendation - finish pngchucklist before exam

1. HW 3 - Due next Wednesday at midnight

1. Exam - Next week Thursday

Exam 1

1. Study guide - Released now under announcements on the website

1. Practice exam - Released by the weekend

1. Exam Review - Next Tuesday during class

Study Tips - redo clickers, hw, and start MP3-PNG (pngchunklist)

Review

transistors allow or disallow current ⇒ Gates combine transistors ⇒ logic + math is implemented with gates

storage

fast, big
slow, small

caching is how we combine these in a smart way

↓

1 and 0. so everything is in 1 and 0's → can use hex to view 1, 0's

Central Processing Unit

Today’s LGs - Have a high level understanding of how we get from C to voltages across hardware.

High level logical steps ➜ voltages across gates ➜ results

logic → C → assembly → machine code → results

Agenda

1. CPU Hardware

~~1~~2. Execution Cycle

~~1~~3. Assembly

~~1~~4. Optimization

Central Processing Unit

Hardware executes simple instructions... in the cpu by moving around voltages following fetch, decode, execute

Fetch - get instruction

Decode - interpret instruction

Execute - execute instruction

Hardware in a CPU

Registers - small storage in CPU (64 bits big each)

General registers - your cpu has 16-32 of these

PC registers - stores the address of the next instruction

instruction registers - holds current instruction

ALU - arithmetic logic unit

- performs math, logic, comparisons

Control Unit - moves bits around to coordinate fetch-decode-execute cycle

What is fed into the ALU by the Control Unit from the Registers?

clicker.cs.illinois.edu

Q2

~Code~
340

A) transistors

B) cache

C) 16 values of voltages

[✓ D) 2 values of voltages]

What hardware does a CPU contain?

clicker.cs.illinois.edu

Q3

~Code~
340

[✓ A) gates]
[✓ B) transistors]
C) RAM
D) SSD

Execution Cycle

Fetch - control unit gets instruction from memory at address held in the PC register

Decode - control unit interprets the instruction

Execute - control unit moves bits around per instructions

PC gets incremented . . . repeat!

What can a CPU execute?

CPU needs Rxplicit instructions on what bits to move where

General Actions a cpu can do

move data = move/load/store

compute = add/sub/mul

control flow = jmp

ISA - Instruction Set Architecture

- specifics on what instructions the cpu can run
- could be different on different computers

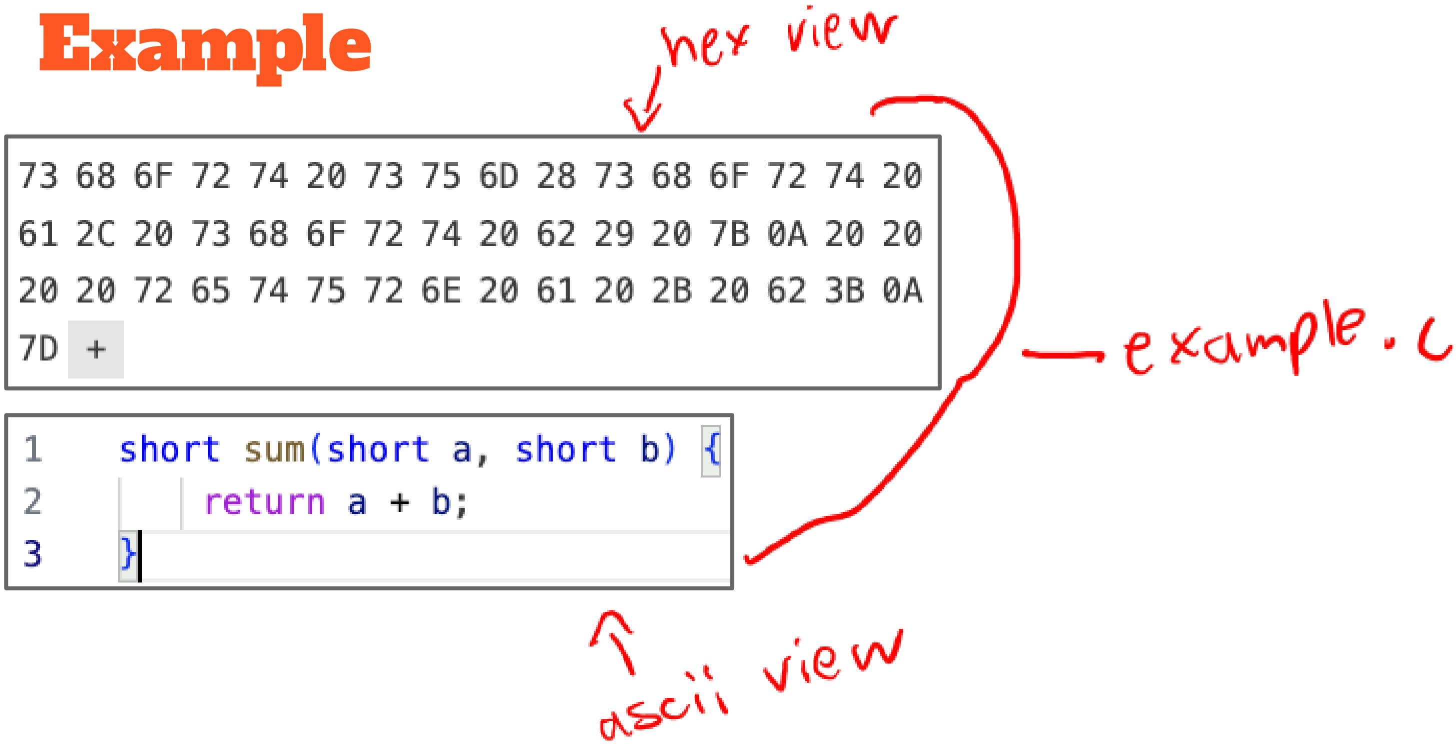
instruction arg arg ....

value

register name

address

Example



```
73 68 6F 72 74 20 73 75 6D 28 73 68 6F 72 74 20
61 2C 20 73 68 6F 72 74 20 62 29 20 7B 0A 20 20
20 20 72 65 74 75 72 6E 20 61 20 2B 20 62 3B 0A
7D +
```

```c
short sum(short a, short b) {
    return a + b;
}
```

What is ‘s’ in binary?

A) 0x73

B) 73

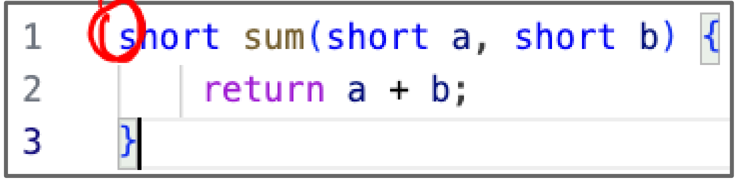
[✓ C) 0b0111 0011]

D) 0b0100 1001

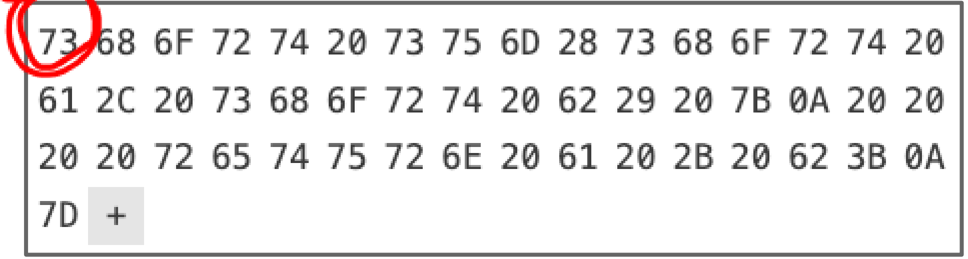
clicker.cs.illinois.edu

Q4

~Code~
340

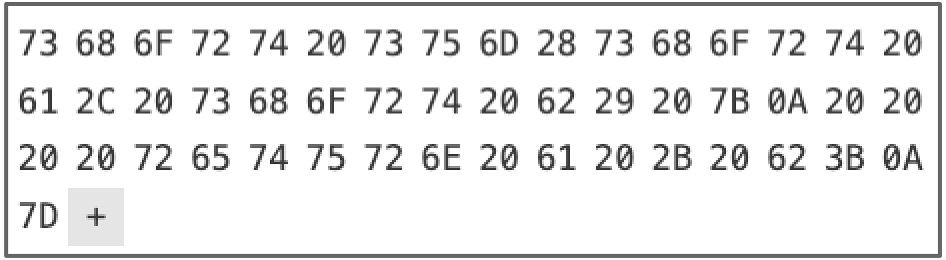


```
short sum(short a, short b) {
    return a + b;
}
```

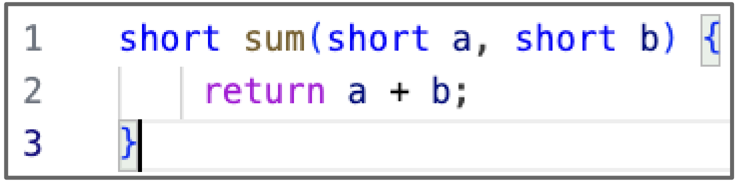


```
73 68 6F 72 74 20 73 75 6D 28 73 68 6F 72 74 20
61 2C 20 73 68 6F 72 74 20 62 29 20 7B 0A 20 20
20 20 72 65 74 75 72 6E 20 61 20 2B 20 62 3B 0A
7D
```

C to Machine Code



```
73 68 6F 72 74 20 73 75 6D 28 73 68 6F 72 74 20
61 2C 20 73 68 6F 72 74 20 62 29 20 7B 0A 20 20
20 20 72 65 74 75 72 6E 20 61 20 2B 20 62 3B 0A
7D
```

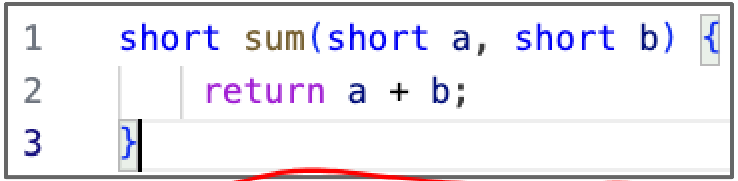


```c
short sum(short a, short b) {
    return a + b;
}
```

= ?

cpu needs to know what to put where?

Simplified Assembly and Machine Code



```
short sum(short a, short b) {
    return a + b;
}
```

=> assembly => machine code

01 F7 89 F8 C3

01 = add instruction

F7 = 11 110 111

mode

reg #1

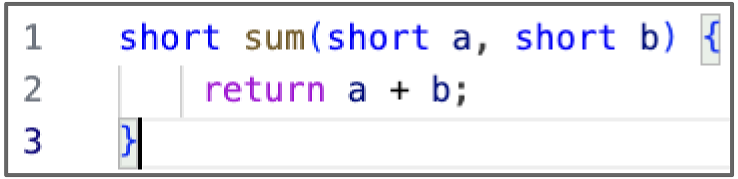
reg #2

AMD64-ISA

edi = edi + esi

```
sum:
    addl %esi %edi
    movl %edi %eax
    ret
```

Example #2



```c
short sum(short a, short b) {
    return a + b;
}
```

arm64-ISA

```
Sum:
    add w0, w0, w1
    ret
```

w0 = w0 + w1

0B 01 00 0B

00001011 0000	0001 0	000 00	00 0000	0 1011
opcode add	reg #1	shift	reg #2	reg #3

Big Picture Takeaways

C code → assembly → machine code

⇓

CPU executes

assembly

- move
- compute
- control flow

ISA can vary between computers... describes 'the machine' code your computer understands

I am guaranteed to be able to run a C executable I made on my computer on your computer.

clicker.cs.illinois.edu

Q5

~Code~
340

A) True

[✓ B) False]

The process goes, C code → XXXXX → machine code

clicker.cs.illinois.edu

Q6

~Code~
340

A) Hexadecimal

B) Binary

C) Decimal

[✓ D) Assembly]

I can view bytes in which of the following formats...

clicker.cs.illinois.edu

Q7

~Code~
340

[✓ A) Hexadecimal]

[✓ B) Binary]

[✓ C) Decimal]

[✓ D) Ascii]

I can view assembly in which of the following formats...

clicker.cs.illinois.edu

Q8

~Code~
340

A) Hexadecimal
B) Binary
C) Decimal
D) Ascii

Optimizations

The compiler is a program so it is assembly the CPU runs.

It is assembly that results in assembly.

Compile one time ← idea: move computation here to save time later

Run many times

If a C program is 200 lines, the related assembly will be guaranteed to also be 200 lines.

clicker.cs.illinois.edu

Q9

~Code~
340

A) True

[✓ B) False]

Little and Big Endian Review

int arr[2] = {400, 500};

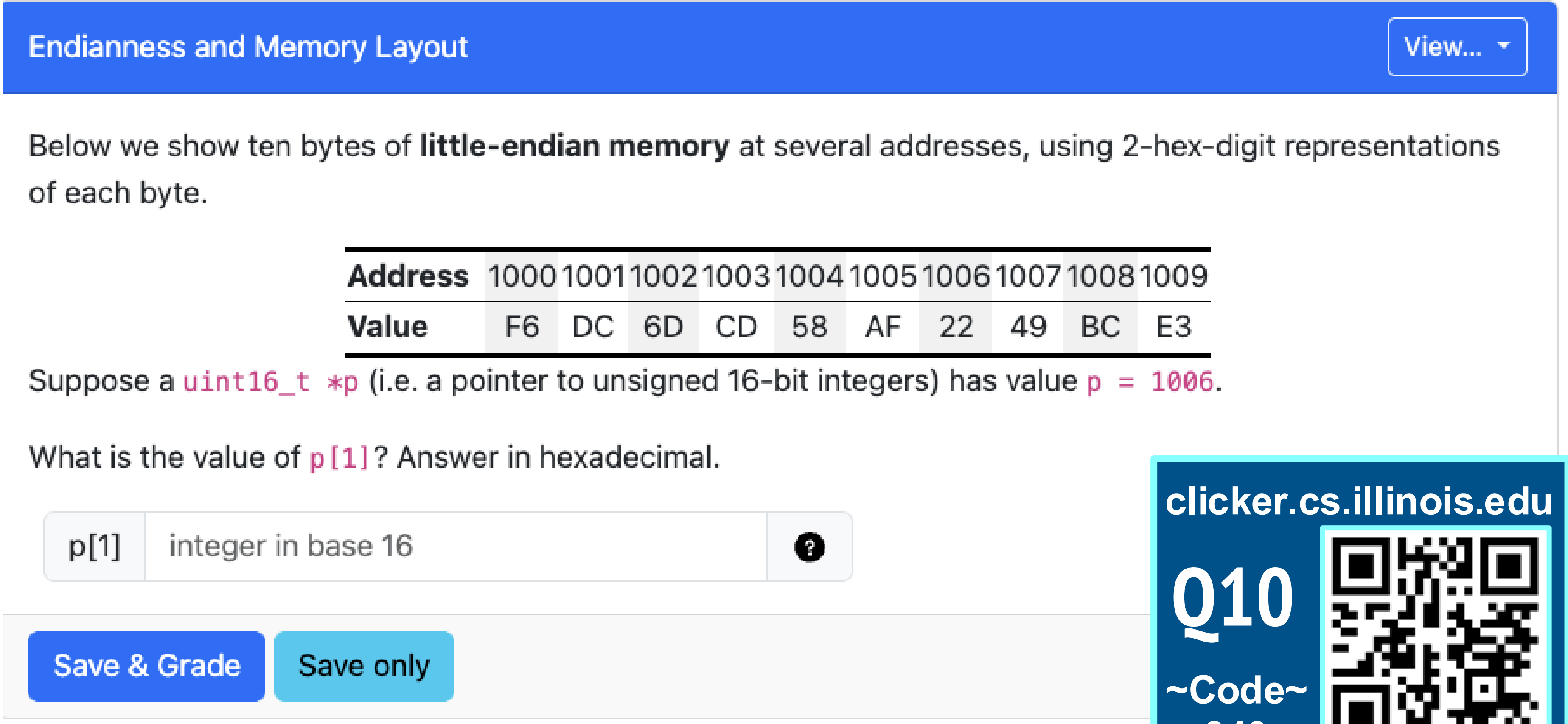
|| 0x190 || 0x1F4

little

0				1				
90	01	00	00	F4	01	00	00	
0x100	101	102	103	0x104	105	106	107	0x108

big

0				1				
00	00	01	90	00	00	01	F4	
0x100	101	102	103	0x104	105	106	107	108



Endianness and Memory Layout

Below we show ten bytes of **little-endian memory** at several addresses, using 2-hex-digit representations of each byte.

Address	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
Value	F6	DC	6D	CD	58	AF	22	49	BC	E3

Suppose a `uint16_t *p` (i.e. a pointer to unsigned 16-bit integers) has value `p = 1006`.

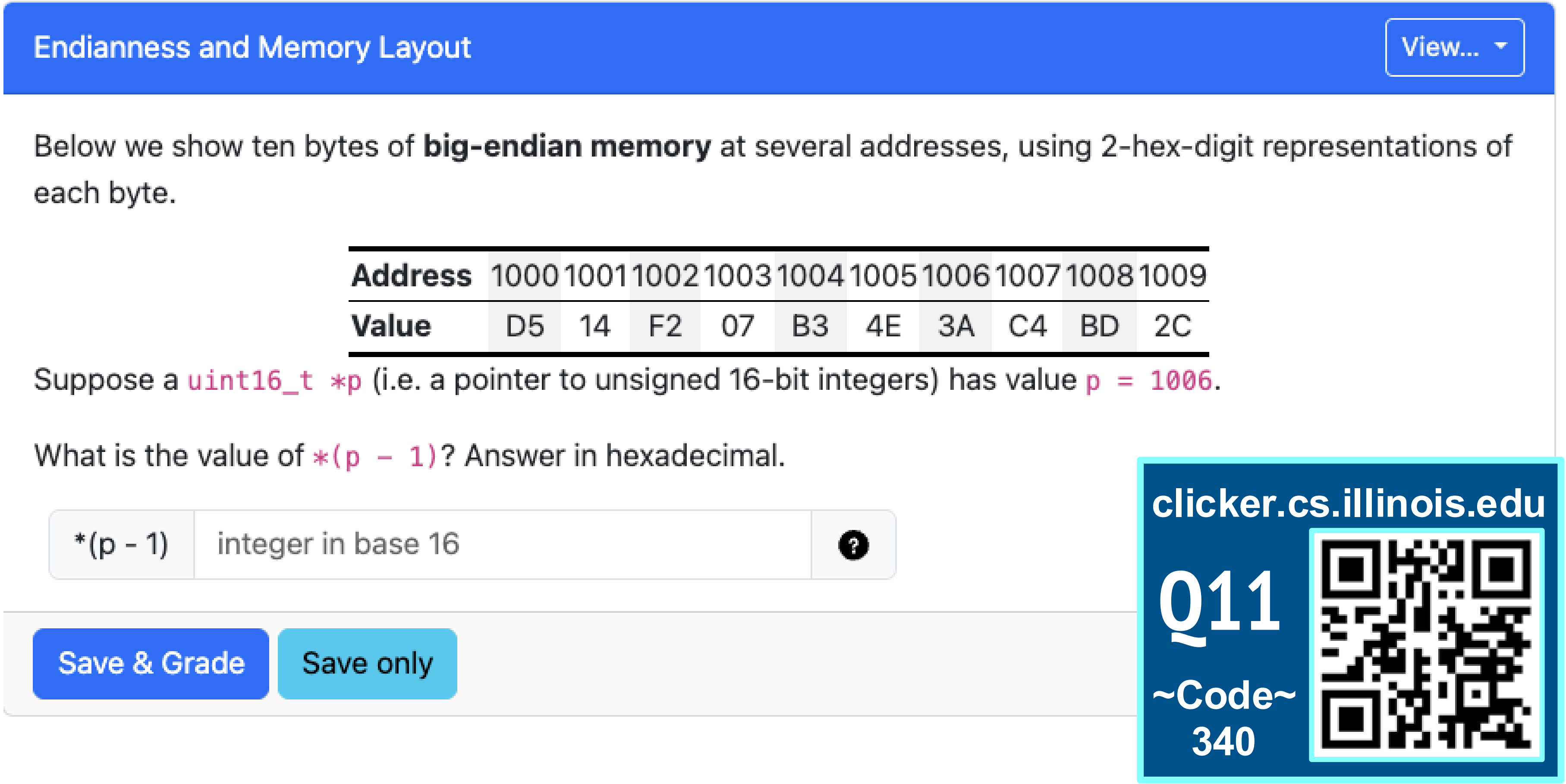
What is the value of `p[1]`? Answer in hexadecimal.

p[1] integer in base 16

clicker.cs.illinois.edu

Q10

~Code~
340



Endianness and Memory Layout

Below we show ten bytes of **big-endian memory** at several addresses, using 2-hex-digit representations of each byte.

Address	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
Value	D5	14	F2	07	B3	4E	3A	C4	BD	2C

Suppose a `uint16_t *p` (i.e. a pointer to unsigned 16-bit integers) has value `p = 1006`.

What is the value of `*(p - 1)`? Answer in hexadecimal.

clicker.cs.illinois.edu

Q11

~Code~
340