香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Tutorial 06: Review for Midterm

Yuhong LIANG

yhliang@cse.cuhk.edu.hk

Outline

- Assignment 1 Solution
- Assignment 2 solution

Assignment 1 Solution – Q1

- Describe the relationship among high-level programming language (e.g. C/C++), assembly language (e.g., MASM), and machine language (or machine code).

❖ High-level languages are converted to assembly language, and the 01 representation of assembly language is machine code. Machine code can directly run on top of the hardware.

Assignment 1 Solution – Q2

- Consider a 32-bit word (B855486B)16:
- (a) What is it if interpreted as a string of characters (according to the below extended ASCII table)?

❖ © U H k

Lec02 p45

- In ASCII encoding scheme, alphanumeric characters, operators, punctuation symbols, and control characters can be represented by 7-bit codes.
  - It is convenient to use an 8-bit byte to represent a character.
    - The code occupies the low-order 7 bits with the high-order bit as 0.

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Assignment 1 Solution – Q2

- Consider a 32-bit word (B855486B)16:
- (b) What is its value in decimal if interpreted as an unsigned integer?

○ B855486B = 1011,1000,0101,0101,0100,1000,0110,1011

❖ $1*2^{31}+1*2^{29}+1*2^{28}+1*2^{27}+1*2^{22}+1*2^{20}+1*2^{18}+1*2^{16}+1*2^{14}+1*2^{11}+1*2^{6}+1*2^{5}+1*2^{3}+1*2^{1}+1*2^{0}=3092596843$

- This vector can represent the decimal value for an unsigned integer $V(B)$ in the range 0 to $2^n - 1$, where

$$V(B) = b_{n-1} \times 2^{n-1} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$$

Lec02 p7

- For example, if $B = (1001)_2, where\ n = 4$

$$V(B) = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (9)_{10}$$

Assignment 1 Solution – Q2

- B855486B = (1011,1000,0101,0101,0100,1000,0110,1011)$_2$
- (c) What is its value in decimal if interpreted as a signed integer using sign-and-magnitude?

❖ (c) $-1*(2^{29}+2^{28}+2^{27}+2^{22}+2^{20}+2^{18}+2^{16}+2^{14}+2^{11}+2^{6}+2^{5}+2^{3}+2^{1}+2^{0})=-945113195$

"Signed" Integer Representation (2/3)

- The leftmost bit (MSB) decides the sign (0: "+", 1: "–").
  - Positive values are identical in all the three systems:
    - *Rule*: Treating the rest bits as an unsigned integer
      - ➢ E.g., +3 is represented by 0011.
  - Negative values have different representations:

① **Sign-and-magnitude** (MSB: sign, other bits: magnitude)
- *Rule*: Changing the MSB from 0 to 1
  - ➢ E.g. –3 is represented by 1011.

```
ex: 0011
     ↓
    1011
```

② **1's-complement**
- *Rule*: Inverting each bit of the positive number
  - ➢ E.g. –3 is obtained by flipping each bit in 0011 to yield 1100.

```
ex: 0011
    ↓↓↓↓
    1100
```

③ **2's-complement**
- *Rule 1*: Subtracting the positive number from the unsigned 2^n
- *Rule 2*: Adding 1 to 1's-complement of that negative number
  - ➢ E.g. –3 is represented by 1101 when applying either rule.

```
ex:
   10000
-)  0011
-------
    1101
```

```
ex:
    1100
+)  0001
-------
    1101
```

Lec02 p12

CSCI2510 Lec02: Number and Character Representation

Assignment 1 Solution – Q2

- B855486B = $(1011,1000,0101,0101,0100,1000,0110,1011)_2$
- (d) What is its value in decimal if interpreted as a signed integer using 1's-complement?

❖ (d) $(1100,0111,1010,1010,1011,0111,1001,0100)_2 = -2^{30}+2^{26}+2^{25}+2^{24}+2^{23}+2^{21}+2^{19}+2^{17}+2^{15}+2^{13}+2^{12}+2^{10}+2^{9}+2^{8}+2^{7}+2^{4}+2^{2} = -1202370452$

Lec02 p12

"Signed" Integer Representation (2/3)

- The leftmost bit (MSB) decides the sign (0: "+", 1: "–").
  - **Positive values** are identical in all the three systems:
    - *Rule*: Treating the rest bits as an unsigned integer
      - E.g., +3 is represented by 0011.
  - **Negative values** have different representations:

① **Sign-and-magnitude** (MSB: sign, other bits: magnitude)
- *Rule*: Changing the MSB from 0 to 1
  - E.g. –3 is represented by 1011.

```
ex: 0011
     ↓
    1011
```

② **1's-complement**
- *Rule*: Inverting each bit of the positive number
  - E.g. –3 is obtained by flipping each bit in 0011 to yield 1100.

```
ex: 0011
    ↓↓↓↓
    1100
```

③ **2's-complement**
- *Rule 1*: Subtracting the positive number from the unsigned 2^n
- *Rule 2*: Adding 1 to 1's-complement of that negative number
  - E.g. –3 is represented by 1101 when applying either rule.

```
ex:
   10000
-)  0011
--------
    1101
```

```
ex:
    1100
+)  0001
--------
    1101
```

Assignment 1 Solution – Q2

- B855486B = $(1011,1000,0101,0101,0100,1000,0110,1011)_2$
- What is its value in decimal if interpreted as a signed integer using 2's-complement?

❖ (e) $(1011,1000,0101,0101,0100,1000,0110,1010)_2$
$(1100,0111,1010,1010,1011,0111,1001,0101)_2$ =$-1*(2^{30}+2^{26}+2^{25}+2^{24}+2^{23}+2^{21}+2^{19}+2^{17}+2^{15}+2^{13}+2^{12}+2^{10}+2^{9}+2^{8}+2^{7}+2^{4}+2^{2}+2^{0})$=-1202370453

Lec02 p12

"Signed" Integer Representation (2/3)

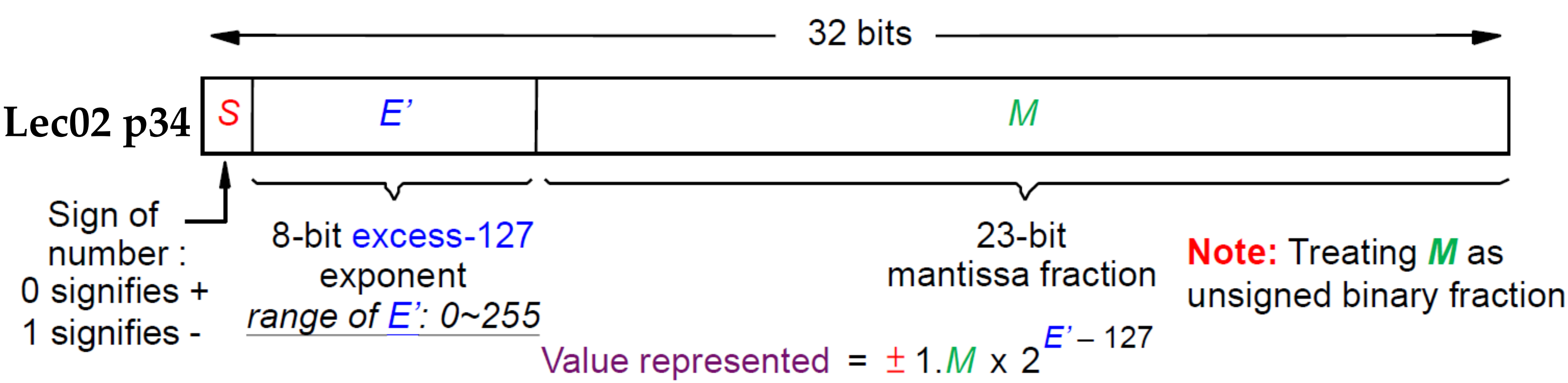
- The leftmost bit (MSB) decides the sign (0: "+", 1: "–").
  - **Positive values** are identical in all the three systems:
    - *Rule*: Treating the rest bits as an unsigned integer
      - E.g., +3 is represented by 0011.
  - **Negative values** have different representations:
  - ① **Sign-and-magnitude** (MSB: sign, other bits: magnitude)
    - *Rule*: Changing the MSB from 0 to 1 ex: 0011 ↓ 1011
      - E.g. –3 is represented by 1011.
  - ② **1's-complement** ex: 0011 ↓↓↓↓ 1100
    - *Rule*: Inverting each bit of the positive number
      - E.g. –3 is obtained by flipping each bit in 0011 to yield 1100.
  - ③ **2's-complement**
    - *Rule 1*: Subtracting the positive number from the unsigned 2^n
    - *Rule 2*: Adding 1 to 1's-complement of that negative number
      - E.g. –3 is represented by 1101 when applying either rule.

```
ex:
 10000
-) 0011
-------
  1101
ex:
  1100
+) 0001
-------
  1101
```

Assignment 1 Solution – Q2

- B855486B = $(1011{,}1000{,}0101{,}0101{,}0100{,}1000{,}0110{,}1011)_2$
- What is its value in decimal if interpreted as a floating-point number using IEEE Standard 754 Single Precision?
- Answer = $-1 * (1.101\ 0101\ 0100\ 1000\ 0110\ 1011_2) * 2^{(011\ 1000\ 0_2)-127}$
- $01110000_2 - 127 = (2^6 + 2^5 + 2^4) - 127$ = -15
- $1.101\ 0101\ 0100\ 1000\ 0110\ 1011_2 = 1.666272521$

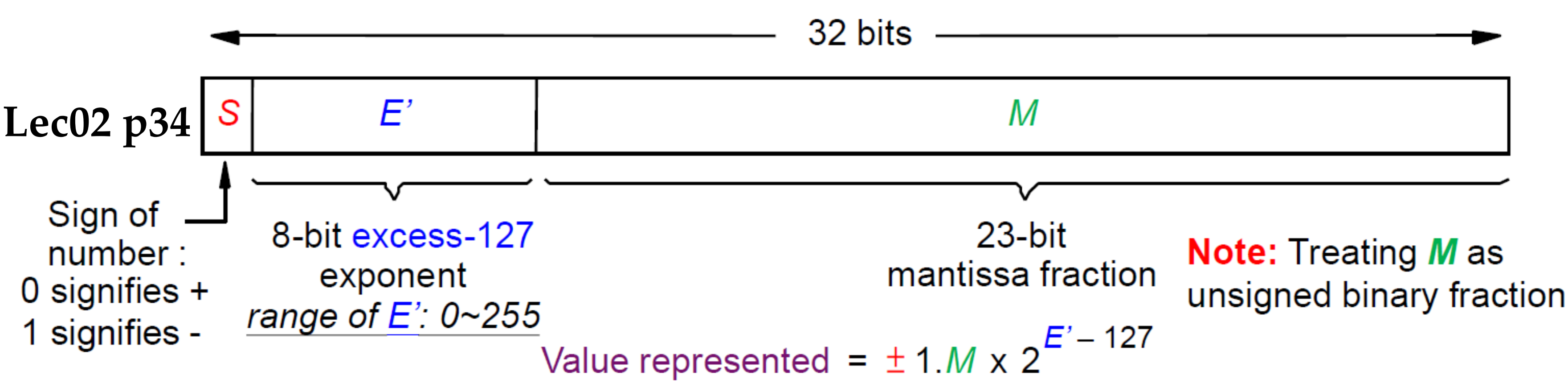
Lec02 p34



(f):

- Answer = $-1 * (1.10101010100100001101011_2) * 2^{(01110000_2)-127} = -1 * 1.666272521 * 2^{-15} = -5.085060183773748 6 * 10^{-05}$

Lec02 p34



Assignment 1 Solution – Q3

Question 3 (30 pts)

Consider a computer system of word size 32 bits and has a main memory system of 3 GB.

(a) How many bits, bytes, and words are there in the memory system?

(b) If the system is byte addressable, what is the minimum number of required bits for memory addresses?

(c) Suppose a 32-bit number $(\texttt{6A738C9E})_{16}$ is stored at word address 200, and a string of characters "**`CSCI2510`**" is stored at word address 204. Please fill the contents of the memory in the following forms when 1) big-endian system and 2) little-endian system are adopted, respectively. *(Note: An N-character string should not be treated as one large multi-byte value, but rather as N single character values. That is, the first character of the string has the smallest byte address, while the last character has the largest byte address.)*

Big endian

	+0	+1	+2	+3
200				
204				
208				

Little endian

	+3	+2	+1	+0
200				
204				
208				

Assignment 1 Solution – Q3

Consider a computer system of word size 32 bits and has a main memory system of 3 GB.

(a) How many bits, bytes, and words are there in the memory system?

- 3GB = 3 Giga Bytes = 3 * 2^{30} Bytes
- Number of words = 3GB / 32 bits = 3GB / 4 B = $3*2^{28}$
- Number of bytes = 3GB / 1 B = $3 * 2^{30}$
- Number of bits = 3GB / 1 bits = 3*8 Gbits / 1 bits = $3 * 2^{33}$

(b) If the system is byte addressable, what is the minimum number of required bits for memory addresses?

- According to (a), $2^{31} <$ the number of byte = $3 * 2^{30} < 2^{32}$. If the system is byte-addressable, the minimum number of required bits for memory address is 32-bit address.

Assignment 1 Solution – Q3

(c) Suppose a 32-bit number **(6A738C9E)**$_{16}$ is stored at word address 200, and a string of characters "**CSCI2510**" is stored at word address 204. Please fill the contents of the memory in the following forms when 1) big-endian system and 2) little-endian system are adopted, respectively. *(Note: An N-character string should not be treated as one large multi-byte value, but rather as N single character values. That is, the first character of the string has the smallest byte address, while the last character has the largest byte address.)*

Big endian

	+0	+1	+2	+3
200	6A	73	8C	9E
204	C	S	C	I
208	2	5	1	0

Little endian

	+3	+2	+1	+0
200	6A	73	8C	9E
204	I	C	S	C
208	0	1	5	2

Two ways to order byte addresses across a word:

- **Big-Endian**: Bytes within a word are ordered left-to-right, and lower byte addresses are used for more significant bytes of a multi-byte data (e.g. Motorola).
- **Little-Endian**: Bytes within a word are ordered right-to-left, and lower byte addresses are used for less significant bytes of a multi-byte data (e.g. Intel).

Lec03 p14

Assignment 2 Solution – Q1

- Suppose that registers R1, R2 and R3 contain the decimal numbers 256, 384 and 512, respectively, and LOC corresponds to the memory address 1024 in decimal. Specify the addressing mode and the effective address (EA) for each of the following operand:

a) R3
   - ❖ Register, EA = R3

b) (R3)
   - ❖ Register indirect, EA = 512

c) (R2, R3)
   - ❖ Base with index,
   EA = 384+512 = 896

d) LOC
   - ❖ Absolute, EA = 1024

e) -128(R2)
   - ❖ Index,
   EA = -128+384 = 256

Addressing Modes: the ways for specifying the contents or locations of instruction operands.

Address Mode	Assembler Syntax	Addressing Function
1) Immediate	$\#Value$	$Operand = Value$
2) Register	Ri	$EA = Ri$
3) Absolute	LOC	$EA = LOC$
4) Register indirect	(Ri)	$EA = [Ri]$
5) Index	$X(Ri)$	$EA = [Ri] + X$
6) Base with index	(Ri, Rj)	$EA = [Ri] + [Rj]$

$Value$: a signed number
EA: the effective address of a register or a memory location
X: an index value

CI2510 Lec04: Machine Instructions 12

Lec04 p12

Assignment 2 Solution – Q2

- Given two 4-bit registers R1 and R2 storing signed integers in 2's-complement format. Please specify the condition flags that will be affected by **SUB R1, R2**.

(a) R1 = $(3)_{10}$ and **R2** = $(4)_{10}$

- ○ $(3)_{10} - (4)_{10} = (0011)2 - (0100)2 = (0011)2 + (1100)_2 = (1111)_2$
- ❖ N = 1, Z = 0, V = 0

(b) R1 = $(1)_{10}$ and **R2** = $(1)_{10}$

- ○ $(1)_{10} - (1)_{10} = (0001)_2 - (0001)_2 = (0001)_2 + (1111)_2 = (\cancel{1}0000)_2$
- ❖ N = 0, Z = 1, V = 0

(c) R1 = $(3)_{10}$ and **R2** = $(-6)_{10}$

- ○ $(3)_{10} – (-6)_{10} = (0011)_2 - (1010)_2 = (0011)_2 + (0110)_2 = (1001)_2$
- ❖ N = 1, Z = 0, V = 1

Lec05 p24

N (negative)	Set to 1 if the result is negative; otherwise, cleared to 0
Z (zero)	Set to 1 if the result is 0; otherwise; otherwise, cleared to 0
V (overflow)	Set to 1 if arithmetic overflow occurs; otherwise, cleared to 0

Lec02 p24

Overflow: The result of an arithmetic operation does not fall within the representable range.

Assignment 2 Solution – Q2

- Given two 4-bit registers R1 and R2 storing signed integers in 2's-complement format. Please specify the condition flags that will be affected by **SUB R1, R2**.

(d) R1 = $(-1)_{10}$ and **R2** = $(1)_{10}$

- ○ $(-1)_{10} - (1)_{10} = (1111)_2 - (0001)_2 = (1111)_2 + (1111)_2 = (\cancel{1}1110)_2$
- ❖ N = 1, Z = 0, V = 0

(e) R1 = $(-7)_{10}$ and **R2** = $(3)_{10}$

- ○ $(-7)_{10} - (3)_{10} = (1001)_2 - (0011)_2 = (1001)_2 + (1101)_2 = (\cancel{1}0110)_2$
- ❖ N = 0, Z = 0, V = 1

(f) R1 = $(7)_{10}$ and **R2** = $(6)_{10}$

- ○ $(7)_{10} - (6)_{10} = (0111)_2 - (0110)_2 = (0111)_2 + (1010)_2 = (\cancel{1}0001)_2$
- ❖ N = 0, Z = 0, V = 0

N (negative)	Set to 1 if the result is negative; otherwise, cleared to 0
Z (zero)	Set to 1 if the result is 0; otherwise; otherwise, cleared to 0
V (overflow)	Set to 1 if arithmetic overflow occurs; otherwise, cleared to 0

Overflow: The result of an arithmetic operation does not fall within the representable range.

Supplement: Please take a look at this weblink to learn how to set carry flag for subtractions if you are interested: http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

Assignment 2 Solution – Q3

- The below program adds up a list of *n* numbers, where the size *n* is stored in memory address **N**, and **NUM1** denotes the memory address of the first number. Rewrite the program to add up this list in a reversed order.

LABEL	OPCODE	OPERAND	COMMENT
	Load	R2, N	Load the size of the list.
	Clear	R3	Initialize sum to 0.
	Move	R4, addr NUM1	Get address of the first number.
LOOP:	Load	R5, (R4)	Get the next number.
	Add	R3, R3, R5	Add this number to sum.
	Add	R4, R4, #4	Increment the pointer to the list.
	Subtract	R2, R2, #1	Decrement the counter.
	Branch_if_[R2]>0	LOOP	Branch back if not finished.
	Store	R3, SUM	Store the final sum.

Assignment 2 Solution – Q3

LABEL	OPCODE	OPERAND	COMMENT
	Load	R2, N	Load the size of the list.
	Imul	R2, R2, #4	Initialize the offset
	Clear	R3	Initialize sum to 0.
	Move	R4, addr NUM1	Get address of the first number.
LOOP:	Subtract	R2, R2, #4	Decrement the offset.
	Load	R5, (R4, R2)	Get the next number.
	Add	R3, R3, R5	Add this number to sum.
	Branch_if_[R2]>0	LOOP	Branch back if not finished.
	Store	R3, SUM	Store the final sum.

Base=R4=addr NUM1, Index=R2=4*N-4

Exercise 1

1.Complete the provided MASM IA-32 assembly program named **stack.asm** to implement a stack.
2.There are **six** "missing lines" in total.

input:
…
cmp ECX, 0; compare content of ECX with 0 (missing line)
je popnum; if content of ECX is equal to 0, then jump to popnum (missing line)
…

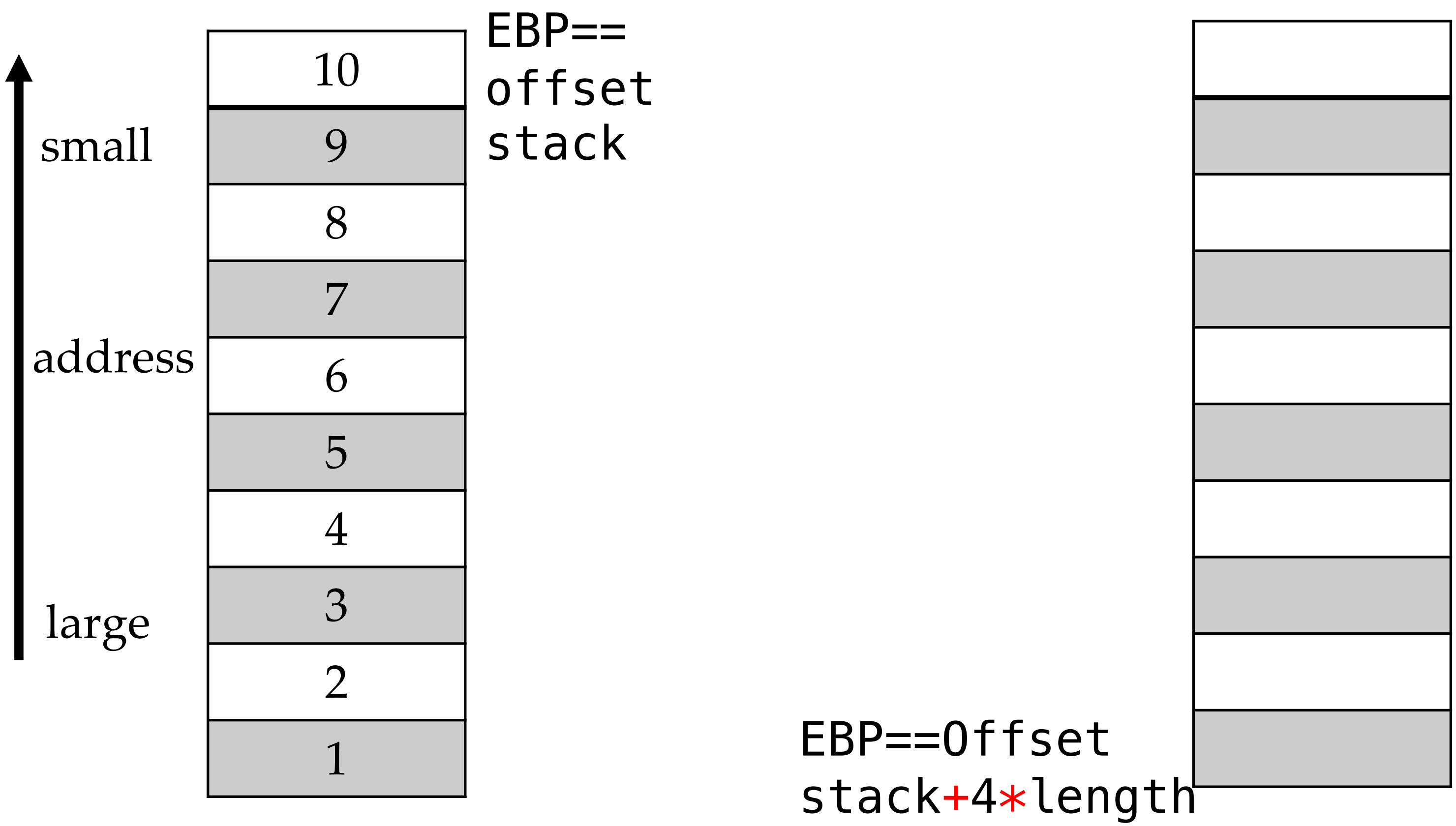
pushnum:
sub EBP, 4 ; decrease the top pointer by 4 (missing line)
mov [EBP], ECX; push the inputnumber into stack in memory (missing line)
jmp input

popnum:
mov ECX, [EBP] ; get the top data of in the stack in memory, and load it to ECX (missing line)
invoke crt_printf, addr outputFormatForPop, ECX ; print out the top data
add EBP, 4 ; increase the top buffer by 4 (missing line)
jmp input

Exercise 2 (1/2)

Possible error 1: Push a number into the stack
when the stack is full.

Possible error 2: Pop a number from the stack when the stack is empty.



Exercise 2 (2/2)

Possible error 1: Push a number into the stack when the stack is full.

EBP==offset stack

```
pushnum:
cmp EBP, offset stack; compare top pointer with the stack smallest address to see if the stack is full
je pusherror ; if stack is full, jump to pusherror
sub EBP, 4 ; decrease the top pointer by 4 (missing line)
mov [EBP], ECX; push the inputnumber into stack in memory (missing line)
jmp input
```

Possible error 2: Pop a number from the stack when the stack is empty.

EBP==Offset stack+4*length

```
popnum:
mov EAX, 4
imul EAX, stacklength
add EAX, offset stack ; compute the largest memory address of stack
cmp EBP, EAX; compare with the largest memory address of stack to see if the stack is empty
je poperror; if the stack is empty, jump to poperror
mov ECX, [EBP] ; get the top data of in the stack in memory, and load it to ECX (missing line)
invoke crt_printf, addr outputFormatForPop, ECX ; print out the top data
add EBP, 4 ; increase the top buffer by 4 (missing line)
jmp input
```

Exercise 3 (1/2)

`gettop`: Print out the number on the top of the stack without popping it.

Remember to add the entrance

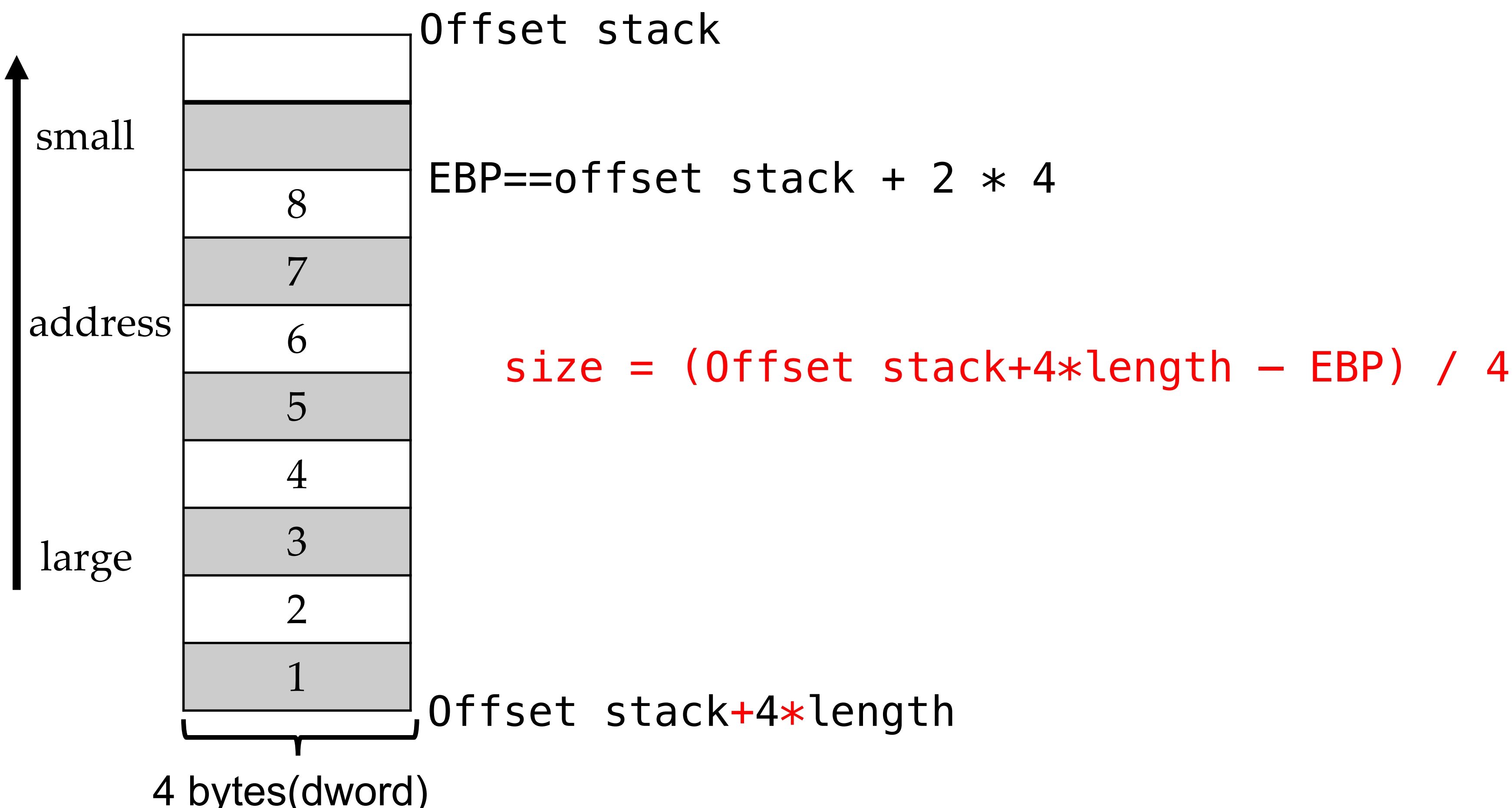
cmp ECX, -2; compare content of ECX with -2
je getsize ; jump to print out the size of numbers that have been pushed into the stack

Similar to popnum without doing the addition

```
gettop:
mov EAX, 4
imul EAX, stacklength
add EAX, offset stack ; compute the largest memory address of stack
cmp EBP, EAX ; see if the stack is empty
je isempty; if the stack is empty jump to isempty
mov ECX, [EBP] ; get the top data of in the stack in memory, and load it to ECX
invoke crt_printf, addr outputFormatForTop, ECX ; print out the top data
jmp input
```

Exercise 3 (2/3)

`getsize`: Print out the size of numbers that **have been pushed** into the stack.



Exercise 3 (3/3)

`getsize`: Print out the size of numbers that **have been pushed** into the stack.

Input:

Remember to add the entrance

…
cmp ECX, -2; compare content of ECX with -2
je getsize ; jump to print out the size of numbers that have been pushed into the stack
…

```
getsize:
mov EBX, EBP ; load the stack length to EBX ; get the top pointer
mov EAX, 4
imul EAX, stacklength
add EAX, offset stack ; compute the largest memory address of stack
sub EAX, EBX  ; get the offset
mov EBX, 4
mov EDX, 0
idiv EBX ; offset / 4 to see the size
invoke crt_printf, addr outputFormatForPrintSize, EAX ; print out the size
jmp input
```

EAX = (Offset stack+4*length – EBP)
Size = EAX/4

Summary

- Assignment 1 Solution
- Assignment 2 Solution