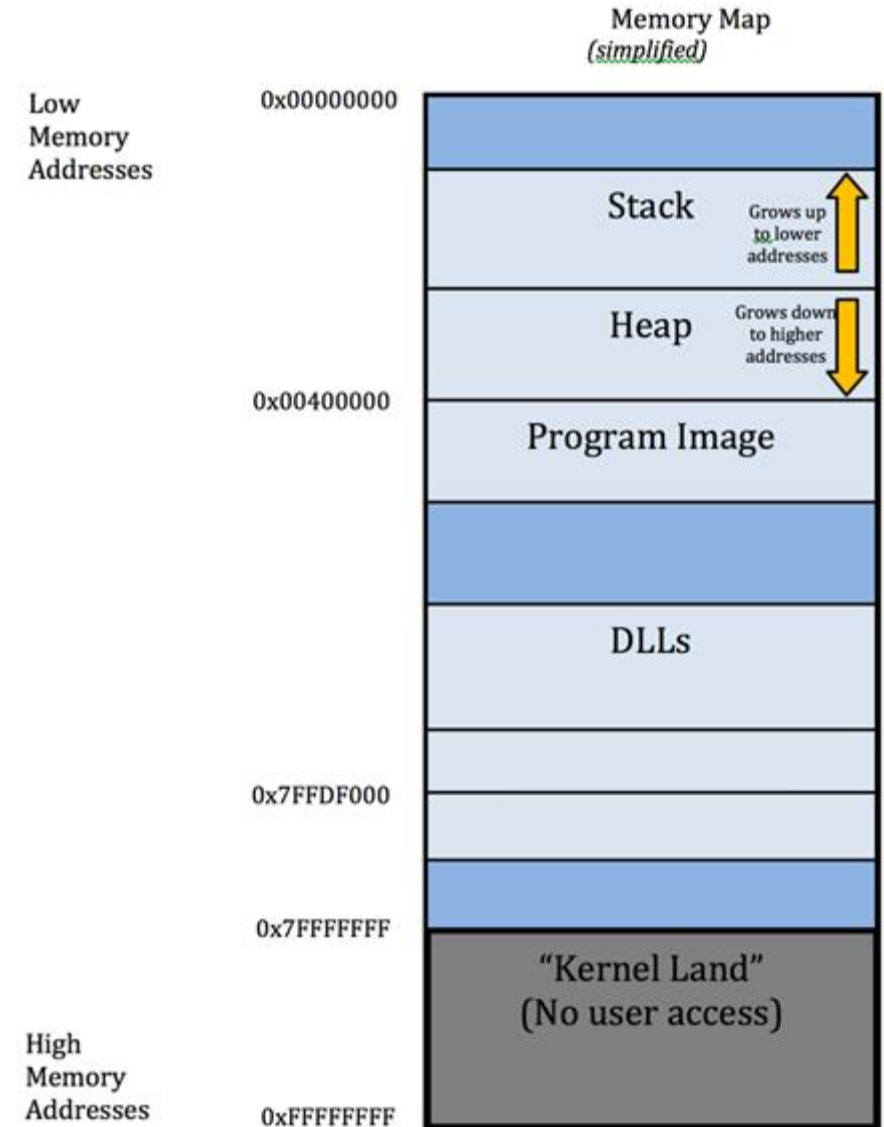


### Sovellukset ja käyttöjärjestelmä

- Tietokoneohjelmat rakentuvat muistissa olevasta datasta ja ohjelmakoodista.
- Ilman dataa ohjelmat eivät ole hyödyllisiä. Esimerkkinä www-selain: Selain käsittelee tietoa, joka haetaan järjestelmän tarjoamaa verkkoyhteyttä käyttäen. Näin näytölle syntyy verkkosivuja, videoita, pelejä, kuvia, musiikkia jne. Tieto tallennetaan johonkin käsittelyn ajaksi. Selaimella täytyy siis olla suoritettava koodi ja data-alue, jota koodi prosessoi.
- Sovelluksen maailma on karkeasti:
  1. muistialue, jossa on suoritettava ohjelmakoodi (program image)
  2. muistialue datalle (heap, keko), jota ohjelmakoodi käsittelee (ns. user land = ohjelman maapalsta...)
  3. muistialue, pino (stack), joka on pikamuisti väliaikaisesti talletettavalle tiedolle. Väliaikainen tässä tarkoittaa pieniä tietomääriä, joita prosessori käsittelee nopeasti, ja joiden elinikä on lyhyt.
  4. muistialue, jota käyttöjärjestelmä hallitsee ohjelman pyytämien käyttöjärjestelmäoperaatioiden varalle (ns. kernel land = käyttöjärjestelmäytimen maapalsta...)
- Käyttöjärjestelmä antaa muistialueet sovellukselle sen käynnistyessä. Alueiden koko voisi olla esim.: user land 2GB ja kernel land 1GB. Pino on pieni, esim. 1 MB.

## Tietokoneen RAM-muisti

- Tietokoneen RAM-muisti koostuu peräkkäin numeroiduista muistipaikoista.
- Kun ohjelma käynnistetään käyttöjärjestelmä varaa ohjelman käyttöön erilaisia muistialueita kääntäjän asettamien vaatimusten mukaan.
- Ohjelmoinnissa tärkeitä muistialueita ovat globaali data-alue, rekisterit, koodialue (program image), vapaan muistin alue (keko, Heap) ja pino (Stack)
- Globaalit muuttujat sijaitsevat globaalilla data-alueella.
- Rekisterit ovat erityislaatuinen osa tietokoneen muistia sillä ne sijaitsevat keskusyksikön (=prosessorin) sisällä. Kaikki tietojenkäsittely tapahtuu rekistereissä. Rekistereillä pidetään kirjaa siitä, missä kohtaa ohjelmaa ollaan menossa ja mikä on tietokoneen tila.
- Koodialue on ohjelmalle varattu osa keskusmuistia, jonne käyttöjärjestelmä lataa ohjelman binääriseen muotoon käännetyt käskyt. Jokainen ohjelmarivi käännetään joukoksi käskyjä ja jokainen näistä käskyistä on omassa muistiosoitteessaan.



# IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

## Muuttuja ja tietokoneen muisti

- Jokainen lokero (=muistipaikka) on yhden tavun kokoinen, ja tavun koko on kahdeksan bittiä

Bitti 7	Bitti 6	Bitti 5	Bitti 4	Bitti 3	Bitti 2	Bitti 1	Bitti 0
$2^7=128$	$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$

- Kun pitää selvittää miten kokonaisluku muutetaan biteksi ja siitä heksaluvuksi käytetään esimerkiksi yllä olevaa taulukkoa
- Esimerkki 1: Muutetaan kokonaisluku 9 ensin biteiksi ja sitten heksaluvuksi
  - Selvitä ensin ylläolevasta taulukosta mitkä bittien arvot antavat yhteenlaskun tulokseksi 9
  - Vastaus: bitti 3 ( $2^3=8$ ) ja bitti 0 ( $2^0=1$ ). Eli lasketaan yhteen  $8+1=9$
  - Oikea bittijono: 0000 1001
  - Oikea heksaluku: 0x09, koska ensimmäiset 4 bittiä (bitit 3->0) antavat arvon 9 ja jälkimmäiset 4 bittiä (bitit 7->4) antavat arvon 0.
- Esimerkki 2: Muutetaan kokonaisluku 17 ensin biteiksi ja sitten heksaluvuksi
  - Vastaus: bitti 4 + bitti 0 =  $16 + 1 = 17$
  - Oikea bittijono: 0001 0001
  - Oikea heksaluku 0x11, koska ensimmäiset 4 bittiä (bitit 3->0) antavat arvon 1 ja jälkimmäiset 4 bittiä (bitit 7->4) antavat myös arvon 1.
  - Neljän bitin (siis bitit 7,6,5,4 ja 3,2,1,0) mukaisesti voidaan saada vain arvot 0-15, eli heksaluvut 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Esimerkki 2: Muutetaan kokonaisluku 171 ensin biteiksi ja sitten heksaluvuksi
  - Vastaus: bitti 7 + bitti 5 + bitti 3 + bitti 1 =  $128 + 32 + 8 + 2 + 1 = 171$
  - Oikea bittijono: 1010 1011
  - Oikea heksaluku 0xAB, koska ensimmäiset 4 bittiä (bitit 3->0) antavat arvon A ja jälkimmäiset 4 bittiä (bitit 7->4) antavat arvon B.

- **Tehtävä 1:** Selvitä bittijono ja heksaluku alla oleviin, kun kokonaisluku on tiedossa
  - Kokonaisluku on 4
  - Kokonaisluku on 9
  - Kokonaisluku on 10
  - Kokonaisluku on 13
  - Kokonaisluku on 24
  - Kokonaisluku on 71
  - Kokonaisluku on 144
  - Kokonaisluku on 255

### Muuttuja ja tietokoneen muisti

- Muuttujan paikka on tietokoneen muistissa, jonne voidaan tallettaa muuttujan arvo.
- Tietokoneen muisti voidaan käsittää lokerikoksi, jossa jokaisella lokerolla on oma numeronsa alkaen nollasta. Nämä numerot ovat muistiosoitteita (esim. 0x22feae).
- Muuttujan **ensimmäinen** muistiosoite voidaan kysyä ohjelmassa osoite-operaattorilla **&** ja muuttujan koko **sizeof()** funktiolla
- Muuttuja varaa yhden tai useamman lokeron joihin muuttujan arvo voidaan tallettaa.
- Kun muuttuja määritellään ohjelmassa (esim. short omaMuuttuja) kääntäjälle kerrotaan minkä tyyppinen muuttuja on kyseessä: kokonaisluku, desimaaliluku, merkki ...
- Muuttujan tyyppistä kääntäjä tietää kuinka paljon sille pitää varata lokeroita muistista ja minkä tyyppistä tietoa muuttujaan voi sijoittaa.
- Jokainen lokero (=muistipaikka) on yhden tavun kokoinen, ja tavun koko on kahdeksan bittiä
- Muistipaikan arvo näytetään ohjelmoitityökalujen debuggreissa heksalukuna esim. (0xAF)
- Heksaluvut (0-F)
  - \* kokonaisluku 0 = heksaluku 0
  - \* kokonaisluku 9 = heksaluku 9
  - \* kokonaisluku 10 = heksaluku A
  - \* kokonaisluku 11 = heksaluku B
  - \* kokonaisluku 12 = heksaluku C
  - \* kokonaisluku 13 = heksaluku D
  - \* kokonaisluku 14 = heksaluku E
  - \* kokonaisluku 15 = heksaluku F

Tyyppi	Koko	Arvoalue
bool	1 tavu	0 (false) tai 1 (true)
char	1 tavu	0 ... 255 (256 erilaista merkkiä)
unsigned short int	2 tavua	0 ... 65535
short int	2 tavua	-32768 ... 32767
int, float	4 tavua	-2147483648 ... 2147483647
double	8 tavua	2.2e-308 ... 1.8e308

### Debuggeri ja debuggaus

- Jos halutaan varmistua ohjelman algoritmien toiminnasta, tutkia toimiiko ohjelma niin kuin halutaan, onko ohjelmassa jotakin vikaa/virheitä tai **halutaan tutkia laitteen muistia**, niin debuggeri on usein helpoin ja paras vaihtoehto tähän. Ennen vanhaan ohjelmaan lisättiin sinne tänne ylimääräisiä tulostuslauseita.
- Debuggerin avulla ohjelma voidaan **pysäyttää** haluttuun kohtaan ja tutkia muistia, katsoa muuttujien arvoja yms.
- Myös ohjelmointia **opetellessa** debuggeri on hyvä väline askeltaa ohjelmaa eteenpäin rivi kerrallaan, ja näin havainnollistaa ohjelman kulkua ja varmistaa sen toimivuus.
- Seuraavan sivun esimerkissä kirjoitetaan uusi ohjelma, ja käydään läpi miten debuggeria käytetään Qt Creatorissa
- Käydään läpi debuggauksessa seuraavia asioita:
  - Debuggauksen aloittaminen ja lopettaminen
  - Keskeytyspisteiden laittaminen
  - Askeltaminen rivi kerrallaan (F10-painike)
  - Muuttujien arvojen tutkiminen

### Debuggeri ja debuggaus

- Kommentoi **main.cpp** tiedostossa oleva koodi kokonaan ja kirjoita alla oleva koodi Qt Creatoriin.
- Käännä ja aja ohjelmaa.

```
#include <iostream>

using namespace std;

int main()
{
    short kokonaislukumuuttuja=0;
    kokonaislukumuuttuja=15;

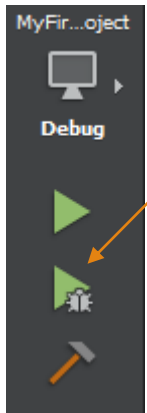
    cout << "kokonaislukumuuttuja arvo: " << kokonaislukumuuttuja<< endl;
    cout << "kokonaislukumuuttuja ensimmäinen muistiosoite: " << &kokonaislukumuuttuja<< endl;
    cout << "kokonaislukumuuttuja koko: " << sizeof(kokonaislukumuuttuja) << endl;

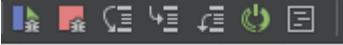



    return 0;
}
```

- Käy läpi seuraavalla sivulla oleva ohje, miten edetään debuggauksessa

## Debuggeri ja debuggaus

- Qt Creator työkalussa debuggeri/debuggaus käynnistetään työkalun vasemmassa alakulmassa olevasta painikkeesta tai päävalikon vaihtoehdosta **Debug->Start Debugging->Start Debugging** tai painamalla F5



- Kun debuggaus käynnistyy ilmestyy työkaluun painikevalikko 
  - Kun työkalussa olevan painikkeen päälle siirtää hiiren osoittimen, niin painikkeen toiminto tulee esille
- Tässä vaiheessa tärkeimmät painikkeet
  -  Lopetetaan debuggaus
  -  Step over (tai näppäin F10). Hypätään ohjelman suorituksessa seuraavaan käskyyn.
  -  Instruction-wise operation mode. Avaa koodin assembly muodossa.



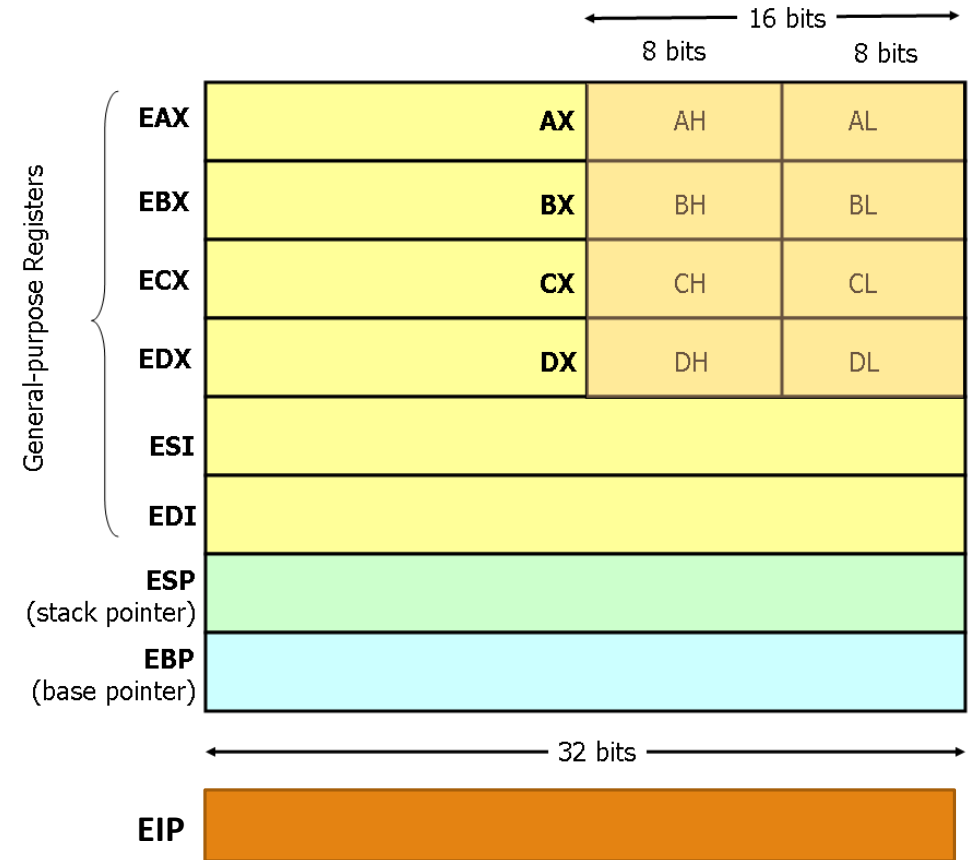
### Debuggeri ja debuggaus

- Aseta keskeytyspiste riville **return 0;** (klikkaa koodirivinumeron vasemmalla puolella. Keskeytyspisteeksi ilmestyy punainen pallo). Keskeytyspisteen saa pois klikkaamalla punaista palloa uudestaan.
- Aloita Debuggaus (Debug->Start Debugging->Start Debugging tai painamalla F5)
- **Kirjoita ylös** aukeavasta ohjelmaikkunasta muuttujan **short kokonaislukumuuttuja** ensimmäinen muistipaikka. *HUOM! short -tyyppiselle muuttujalle varataan keskusmuistista **kaksi muistipaikkaa**, koska **short** tyyppi on määritelty 2:n tavun kokoiseksi.*
- Siirrä hiiren osoitin työkalun ikkunan **Locals and Expressions** (jos ei ikkuna näy valitse Windows->Views-> Locals and Expressions) päälle ja klikkaa hiiren oikeaa painiketta. Valitse aukeavasta valikosta vaihtoehto **Open Memory Editor** ja edelleen aukeavasta valikosta myös vaihtoehto **Open Memory Editor**.
- Kirjoita aukeavaan ikkunaan muuttujan **short kokonaislukumuuttuja** ensimmäisen muistipaikan osoite ja klikkaa OK-painiketta, jonka jälkeen näytölle avautuu muisti-ikkuna. Muuttujan arvo 15 (heksalukuna 0x0F) voidaan lukea muuttujan ensimmäisen muistipaikan kohdalta. Seuraavassa muuttujalle varatussa muistipaikassa arvo on 0x00.
- Muisti-ikkuna suljetaan ikkunan keskellä ylhäällä olevasta rastista. Voit sulkea muisti-ikkunan ja lopettaa debuggauksen.
- Anna muuttujalle ohjelmassa arvot 16, 39, 140, 255 ja muita haluamisia arvoja (kuitenkin max.255) ja debuggaa ohjelmaa.
  - Kirjoita ylös mitä arvoja muuttujan **short kokonaislukumuuttuja** kahteen muistipaikkaan tulee


# IN00BQ93 Laite- ja tuotesuunnittelun syventävät opinnot: ohjelmoinnin jatkokurssi

## Proessorin rekistereistä

- Alla yksinkertaistettu kuva x86-proessorin (32-bit) *rekistereistä*. Rekisterit ovat pieniä datavarastoja CPU:n sisällä, joiden avulla alimman tason toiminnot tehdään. Rekisterit tallentavat mitättömän määrän dataa. Tämän takia ohjelma tarvitsee muistia datan pidempiaikaiseen sijoittamiseen, josta sitä luetaan prosessoriin käsiteltäväksi.
- Rekisterit EAX...EDI ovat yleiskäyttöisiä datavarastoja. ESI (source index) ja EDI (destination index) sisältävät usein osoitteita *heap* alueelta, joista dataa luetaan muihin rekistereihin.
- EBP (base pointer) tallentaa *pinomuistin* alimman muistiosoitteen.
- ESP (stack pointer) puolestaan sisältää *pinon* ylimmän muistiosoitteen, jossa *pinon viimeisin* ('päällimmäinen') data sijaitsee.
- EIP –rekisteri sisältää seuraavaksi suoritettavan käskyn muistiosoitteen. Kun prosessori hyppää seuraavaan komentoon EIP kasvaa automaattisesti yhdellä osoitteella eteenpäin. Näin prosessori pysyy järjestyksessä siitä, missä on menossa.
- Tutustutaan IP (IP= Instruction Pointer, ohjelmaosoitinrekisteri) –rekisteriin debuggerin avulla seuraavalla sivulla.

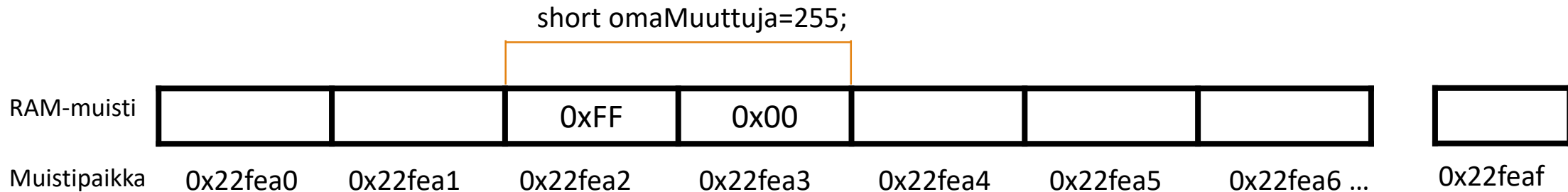


### Debuggeri ja debuggaus

- Aseta keskeytyspisteet riveille
  - \* short kokonaislukumuuttuja=0;
  - \* return 0;
- Aloita Debuggaus
- Aseta rekisterit näkymään päävalikosta Window->Views->Registers
- IP-rekisteri on nimellä **eip**, selvitä sen arvo (=seuraavaksi suoritettavan käskyn muistiosoite)
  - Kun prosessori hyppää seuraavaan komentoon, **eip** kasvaa automaattisesti yhdellä osoitteella eteenpäin. Näin prosessori pysyy järjestyksessä siitä, missä on menossa.
- Aseta debuggerin assembly toiminto päälle klikkaamalla painiketta  **Insturction-wise operation mode**
- Askella (F10) ohjelmaa eteenpäin ja seuraa miten arvo rekisterissä **eip** muuttuu
- Tästä tilasta päästään takaisin C++ koodiin klikkaamalla Insturction-wise operation mode painiketta uudestaan.
- Lopeta Debuggaus

### Muuttuja ja tietokoneen muisti

- Alla olevassa esimerkissä on short tyyppinen muuttuja **omaMuuttuja**, jolle käyttöjärjestelmä varaa 2 tavua muistia.
- Varatut muistipaikat muuttujalle ovat 0x22fea2 ja 0x22fea3
- Muuttujan arvo on 255, joten ensimmäisessä muuttujan muistipaikassa 0x22fea2 on heksalukuarvo 0xFF. Toisessa muistipaikassa arvo on 0x00 (=0)



- Muuttujan ensimmäiseen muistipaikkaan voidaan siis asettaa arvoja 0-255 (0x00 – 0xFF)
- Mitä tehdään kun mennään muuttujassa yli arvon 255? Siitä seuraavalla sivulla!

### Muuttuja ja tietokoneen muisti

- Mitä tehdään kun mennään muuttujassa yli arvon 255? Otetaan käyttöön toinen muistipaikka (bitit 8-15)!
- short muuttuja on kahden muistipaikan kokoinen, joten toinen muistipaikka voidaan ottaa käyttöön.
- short muuttuja voi saada arvoja -32768 ... 32767
- Muuttujan ensimmäinen muistipaikka on siis alla olevan taulukon (bitit 0-7) mukainen, ja näillä biteillä voidaan päästä maksimissaan arvoon 255 (1111 1111, 0xFF)

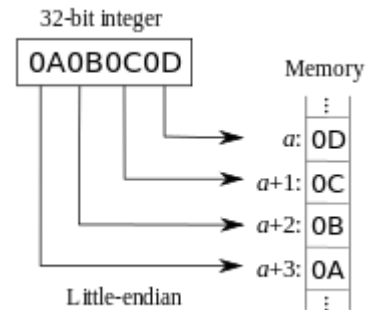
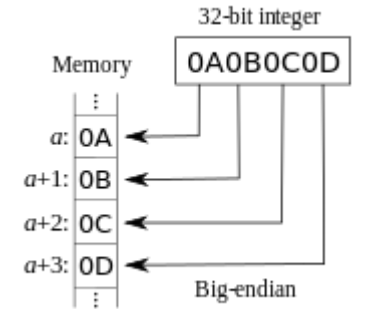
Bitti 7	Bitti 6	Bitti 5	Bitti 4	Bitti 3	Bitti 2	Bitti 1	Bitti 0
$2^7=128$	$2^6=64$	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$

- Muuttujan toinen muistipaikka sisältää bitit ja arvot alla olevan mukaisesti

Bitti 15	Bitti 14	Bitti 13	Bitti 12	Bitti 11	Bitti 10	Bitti 9	Bitti 8
$2^{15}=32768$	$2^{14}=16384$	$2^{13}=8192$	$2^{12}=4096$	$2^{11}=2048$	$2^{10}=1024$	$2^9=512$	$2^8=256$

## Tietokoneen muisti: Tavujärjestys

- Tavujärjestys tietokoneen muistissa tarkoittaa sitä, missä järjestyksessä tietokoneen suoritin käsittelee suurempia kuin yhden tavun pituisia kokonaisuuksia.
- Tavujärjestys muistissa voi olla Big-endian tai Little-endian tyyppinen
- *Big endian* -muodossa bittijonon merkitsevimmät bitit tallentuvat ensin eli alempiin muistiosoitteisiin
- *Little endian* -muodossa bittijonon vähiten merkitsevät bitit tallentuvat ensin
- Anna ohjelmassa kokonaislukumuuttujalle arvo 2571
  - $2048 + 512 + 8 + 2 + 1 = 2571$  eli bittijono = 0000 1010 0000 1011 ja heksaluku = 0x0A0B
  - (0000 1010 = 0x0A ja 0000 1011 = 0x0B)
  - Koska käytössä **little endian** tavujärjestys on muuttujan muistipaikoissa arvot järjestyksessä 0B ja 0A.
- Aseta keskeytyspiste riville **return 0;** ja debuggaa ohjelmaa



### Tietokoneen muisti: Muuttujan ylivuoto ja alivuoto

- Ylivuodossa luvun arvoalueen yläraja ylitetään, jolloin “pyörähdetään” alarajan kautta negatiivisten lukujen puolelle.
- Alivuoto tapahtuu, kun arvoalueen alaraja alitetaan.
- Yli- ja alivuoto voi syntyä aritmeettisen operaation tuloksena tai huonosti tehdyn tyyppimuunnoksen vuoksi.
- ***Yli- ja alivuodot eivät aiheuta ajonaikaista virhettä!***
- Anna ohjelmassa kokonaislukumuuttujalle arvo 32768 ja debuggaa ohjelmaa, mitä tapahtuu muuttujan arvolle?  
\* Tulee muuttujan **ylivuoto**, koska **short** tyyppisen muuttujan arvoalue = -32768 ... 32767
- Aseta keskeytyspiste riville **return 0;** ja debuggaa ohjelmaa.
- Kokeile myös arvolla -32769

### Tehtävä 2: Muistipaikkojen bittijonot

- Selvitä taulukossa short muuttujan muistipaikkojen bittijonot ja heksaluvut, kun kokonaisluku on tiedossa.
- Taulukossa esimerkkinä luvut 255 ja 256
- Testaa lukuja myös ohjelmassa ja tarkista muisti-ikkunasta oikeat heksa-arvot

Kokonaisluku	Ensimmäisen muistipaikan bittijono	Ensimmäisen muistipaikan heksaluku	Toisen muistipaikan bittijono	Toisen muistipaikan heksaluku
255	1111 1111	0xFF	0000 0000	0x00
256	0000 0000	0x00	0000 0001	0x01
257				
4352				
4360				
17152				
26101				
32767				



### Harjoituksia

- Lisää ohjelmaan alla olevia osia, debuggaa ohjelmaa ja tarkastele muisti-ikkunasta muuttujien muistipaikkojen arvoja.
- Sivulla <http://gregstoll.dyndns.org/~gregstoll/floattohex/> voit nopeasti muuttaa lukuja **float->hex** ja **double->hex**.
- Sivulla <http://www.binaryhexconverter.com/decimal-to-hex-converter> voit nopeasti muuttaa lukuja **int->hex**
- Nämä nopeuttavat muistin tarkastelua.
  - Lisää ohjelmaan muuttuja tyyppiä **unsigned short int** ja aseta muuttujan alkuarvo, tulosta muuttujan arvo, ensimmäinen muistiosoite ja koko. Kokeile myös arvoilla 65535 ja 65536.
  - Lisää ohjelmaan muuttuja tyyppiä **int** ja aseta muuttujan alkuarvo, tulosta muuttujan arvo, ensimmäinen muistiosoite ja koko. Kokeile myös arvoa 2147483647.
  - Lisää ohjelmaan muuttuja tyyppiä **double** ja aseta muuttujan alkuarvoksi 20.15, tulosta muuttujan arvo, ensimmäinen muistiosoite ja koko.
  - Lisää ohjelmaan muuttuja tyyppiä **float** ja aseta muuttujan alkuarvoksi 30.95, tulosta muuttujan arvo, ensimmäinen muistiosoite ja koko.
  - Anna muuttujatyyppille **int** negatiivisia arvoja ja tutki muistia.