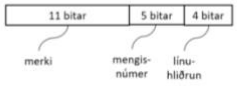
1. Segjum að tölvukerfi hafi vistföng sem eru 20 bitar. Það hefur skyndiminni sem er 2048 bæti að stærð, með línustærð (*block size*, *B*) 16 bæti og það hefur 32 mengi (*S*).

a. Hver er vídd (*associativity*) skyndiminnisins (þ.e. hversu mörg sæti eru fyrir línur í hverju mengi, kallað *E* í bók og glærum)? Sýnið útreikning og útskýrið.
b. Sýnið skiptingu vistfanga í merkishluta (*tag*), mengisnúmer (*set index*) og línuhliðrun (*block offset*). Rökstyðjið hvern hluta.
c. Lesa á 4-bæta orðið með 20-bita vistfangið 0x0B1E4. Útskýrið hvar það getur lent í skyndiminninu sem lýst er að ofan.
d. Útskýrið með reikningi hverskonar vistföng geta lent í mengi númer 23 (= 0x17) í skyndiminninu (þ.e. á hvaða formi eru vistföng sem lenda í þessu mengi)?

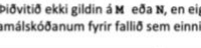
a. Finna vídd skyndiminnisins. Höfum að stærð skyndiminnisins er 2048, sem er $32*E*16$. Getum skrifað þetta $2^{20} = 2^5*2^{*}2^4$. Þá er augljóst að $e=2$ og $E = 2^2 = 4$. Svo þetta er 4-vítt skyndiminni.

b. Við vitum að neðstu bitar vistfangsins eru línuhliðrun og hér er hún 4 bitar (því línustærðin er 16 bæti). Næstu bitar þar fyrir ofan eru mengisnúmerið og hér er það 5 bitar (því fjöldi mengja er 32). Restin af bitunum, þ.e. 20 − 5 − 4 = 11 eru merkið



c. Lesa 4-bita orð með vistfang 0x0B1E4. Skrifum það í bitum sem 0000 1011 0001 1110 0100. Neðstu 4 bitarnir eru 0100, svo línuhliðrunin er 4. Næstu 5 bitar eru 11110 = 0x1E = 30. Merkið er efstu 11 bitarnir: 000 0101 1000 = 0x058. Línan með orðinu lendir því í mengi 30, í einhverju af fjórum sætunum sem eru þar. Merkið 0x058 þarf að passa til að hún sér þar. Við náum svo í 4-bæta orðið í sæti 4 innan línunnar.

d. Skoða hvernig vistföng lenda í mengi númer 23 = 0x17 í skyndiminninu. Þá er mengisnúmerið 0x17 og fimm bitarnir sem tákna það hafa gildið 10111. Aðrir bitar vistfangsins skipta ekki máli. Vistföngin líta því svona út



2. [Próf '21] Hér fyrir neðan er C fall sem afritar eitt stak á milli tveggja tvíviðra fylkja. Fylkineru víðvær og er fylkið **a** afstærðinni **MxN**, en **b** er af stærðinni **NxM**. Þíðvitið ekki gildin á **M** eða **N**, en eigið að geta fundið þau út frá smalamálskóðanum fyrir fallið sem einnig er gefinn.

```
long int a[M][N];
long int b[N][M];

void afrit( int i, int j ) {
    a[i][j] = b[j][i];
}
```

```
afrit:
        movslq  %edi, %rdi
        movslq  %esi, %rsi
        leaq    0(,%rsi,8), %rax
        subq    %rsi, %rax
        addq    %rdi, %rax
        movq    b(,%rax,8), %rdx
        leaq    (%rdi,%rdi,4), %rax
        leaq    (%rdi,%rax,2), %rax
        addq    %rax, %rsi
        movq    %rdx, a(,%rsi,8)
        ret
```

a. Hver eru gildin á **M** og **N**? Rökstyðjið svörin út frá skipununum í smalamálskóðanum.
b. Ef bæði fylkin hefðu verið skilgreind með stærðina **MxN** hefði þá verið hægt að finna gildið á bæði **M** og **N** (eða annað hvort þeirra) út frá smalamálskóðanum? Rökstyðjið svar ykkar.

2. Finna út gildin á **M** og **N** þar sem fylkin eru **a[M][N]** og **b[N][M]**.

a. Þar sem tvíviðið fylki eru geymd í minni eftir línum, þá þarf að vita fjölda dálka í fylkinu til þess að geta reiknað staðsetningu staks (*i*, *j*) í því. Staðsetningin er *i*dálkar + j*.

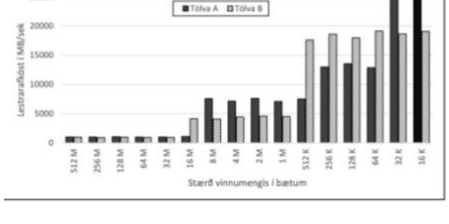
Við erum að afrita stak **b[j][i]** yfir í **a[i][j]**. Til að finna **b[j][i]** þarf að reikna **j*M+i** og til að finna staðsetningu **a[i][j]** þarf að reikna **i*N+j**. Við þurfum að finna þessa tvo útreikninga í smalamálskóðanum. Í kóðanum er breytan **i** í gisti **%rdi** og breytan **j** í gisti **%rsi**. Til að reikna út staðsetningu **b[j][i]** eru notaðar skipanirnar þrjár hér fyrir neðan

Sjáum að þarna fær gistið **%rax** gildið **7*j+i**, svo við vitum að **M = 7**. Skoðum nú útreikning á staðsetningu **a[i][j]**. Þar eru líka notaðar þrjár skipanir

```
leaq (%rdi,%rdi,4), %rax    # rax = 5*i
leaq (%rdi,%rax,2), %rax    # rax = i+2*5*i = 11*i
addq %rax, %rsi             # rsi = 11*i+j
```

Hér fær gistið **%rsi** gildið **11*i+j**, svo **N = 11**.

b. Ef bæði fylkin hefðu haft viddina **MxN** þá hefðu formúlurnar verið **i*N+j** (fyrir **b[j][i]** og **j*N+i** (fyrir **a[i][j]**). Gildið **M** hefði þá ekki komið fyrir í smalamálskóðanum og við hefðum því ekki getað fundið gildi þess út frá kóðanum.



a. Hversu mörg lög (*levels*) af skyndiminni eru í tölvu A og hversu stórt má áætla að hvert þeirra sé? Rökstyðjið svarið!
b. Hversu mörg lög (*levels*) af skyndiminni eru í tölvu B og hversu stórt má áætla að hvert þeirra sé? Rökstyðjið svarið!
c. Tiltekið forrit notar tætitöflu (*hash table*) nokkuð mikið. Taflan hefur 50.000 stök, sem hvert er 8 bæti að stærð. Hvor tölvan væri hagkvæmari fyrir þetta forrit? Rökstyðjið.
d. Breytist svar ykkar við c-lið ef tætitaflan stækkar, t.d. tvöfaldast eða þrefaldast? Rökstyðjið.
e. Er hægt að segja eitthvað um uppsetningu skyndiminnanna (þ.e. línustærð, vídd, eða fjölda mengja) í tölvunum tveimur út frá þessum súluritum? Rökstyðjið.

a. Út frá myndinni má áætla að Tölva A hafi þrjú lög af skyndiminni auk aðalminnis. Efsta lagið, L1 er u.þ.b. 32KB (tvær súlur lengst til hægri). L2 er u.þ.b. 256KB (næstu 3 súlur frá hægri). Loks er L3 u.þ.b. 8MB (næstu 5 súlur). Restin af súlunum fyrir A er aðalminnið.
b. Tölva B virðist hafa tvö lög af skyndiminni. L1 er um 512KB (fyrstu 6 súlurnar frá hægri) og L2 er um 16MB (næstu 5 súlur frá hægri). Restin af súlunum er svo aðalminnið.
c. Gefin er tætitafla með 50.000 stökum, sem hvert er 8 bæti. Stærð hennar er þá 50.000*8 = 400.000 bæti. Hún ætti því að komast öll fyrir í L1 skyndiminni tölvu B og þá væru lestrarafköstin um 18.000MB/sek. Í tölvu A yrðu afköstin væntanlega ekki nema um 7500MB/sek, því taflan kemst ekki öll í L2 skyndiminni í tölvu A.
d. Ef taflan stækkar mikið, t.d. fer upp í 100.000 stök (þ.e. 800.000 bæti), þá kemst hún ekki lengur öll fyrir í L1 skyndiminni á tölvu B og þá yrðu lestrarafköstin ekki nema 4000MB/sek. Afköstin yrðu áfram 7500MB/sek á tölvu A, sem væri þá orðin hagkvæmari.
e. Þessi súlurit gefa okkur engar upplýsingar um skipulag skyndiminnanna (þ.e. línustærð, vídd eða fjölda mengja). Það vantar fleiri upplýsingar til þess.

1. Hér fyrir neðan er kóðabútur í C. Sýnið gildið á breytunni **x** eftir hverja línu og **útskýrið í hvert sinn** í nokkrum orðum hvers vegna **x** hefur þetta gildi.

```
int x = 3, y;
x += y = 5;
x == (y = 3);
x = y == 2;
x = y == 2 ? y << 1 : y >> 1;
```

```
int x = 3, y;
```
x er 3 vegna gildisveitingar (**y** er óupphafsstillt)

```
x += y = 5;
```
x er 8. **y** fær fyrst gildið 5 og því gildi síðan bætt við **x** (3+5)

```
x == (y = 3);
```
x er áfram 8. **y** fær gildið 3 og 3 borið saman við 8 (en ekkert gert með útkomuna úr samanburðinum)

```
x = y == 2;
```
x er 0. Fyrst er **y** borið saman við 2 (**y** inniheldur 3) og útkoma þess sett í **x**. Útkoman er 0 því samanburðurinn er ósannur.

```
x = y == 2 ? y << 1 : y >> 1;
```
x er 1. **y** (sem er 3) er borið saman við 2. Það er ósatt, svo gildið úr **if**-segðinni er: 3 hliðrað um eitt sæti til hægri, sem er 1. Það gildi er sett í **x**.

4. Hér til hliðar er bútur úr C forriti sem vinnur með heiltölufylkið **a[3][64]**. Það eigið að greina hegðun skyndiminnis í kóðanum. Gerið ráð fyrir því að hvert stak í fylkinu sé 4 bæti (því **sizeof(int)** = 4). Gerið einnig ráð fyrir því að eini minniaðgangurinn í kóðanum sé í fylkið **a**. Allar aðrar breytur eru í gistum. Gerið ráð fyrir að skyndiminnið byrji "tómt", þ.e. enginn hluti fylkisins **a** er í skyndiminninu í upphafi. Við megið líka gefa ykkur að upphafsvistfang fylkisins **a** sé **0x0**.

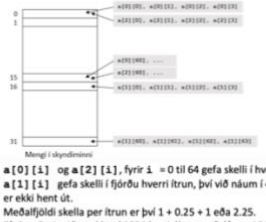
```
int a[3][64];
int i;
int sum = 0;
for (i = 0; i < 64; i++)
    sum = a[0][i]*a[1][i]*a[2][i];
```

a. Gerið ráð fyrir því að skyndiminni noti beina vörpun (*direct-mapped*), stærð þess sé 512 bæti og að línustærðin (*block size*) sé 16 bæti. Hversu mörg mengi eru í þessu skyndiminni?
b. Gefni kóðinn vinnur með allar þrjár línur fylkisins í hverri ítrun. Hvernig varpast stök hverrar línu fylkisins í mengi í skyndiminninu? Með öðrum orðum, í hvaða mengi lenda stök **a[0][0]**, **a[1][0]** og **a[2][0]**, hvar lenda **a[0][1]**, **a[1][1]** og **a[2][1]**, o.s.frv.
c. Það eru þrír minnisaðgangar í hverri ítrun. Hversu oft er hver þeirra skellur? Reiknið út frá því meðalfjölda skella (*miss*) per ítrun í kóðanum (svipað og sýnidæmin með fylkjamargföldun á glærum).
d. Hvernig breytist útreikningur ykkar á c-lið ef skyndiminnið stækkar í 1024 bæti og línustærðin helst áfram 16 bæti? Rökstyðið svar ykkar.

4. Gefinn forritsbútur í C, sem vinnur með **int**-fylkið **a[3][64]**. Greina hegðun skyndiminnis.

a. Skyndiminnið er 512 bæti, með línustærð (*B*) 16 bæti og notar beina vörpun (*E*=1). Finnum fjölda mengja með jöfnunni 512 = S*1*16. Svo S = 32.
b. Fylkið a hefur 3 línur (rows) og 64 dálka (*columns*). Stök þess eru í röð eftir línum og það komast 4 stök fyrir í hverri skyndiminnislínu (blokk). Fyrstu 4 stökin **a[0][0]**, **a[0][1]**, **a[0][2]** og **a[0][3]** lenda því öll í mengi 0 í skyndiminninu. Næstu 4 lenda í mengi 1, o.s.frv. Fyrsta lína fylkisins fyllir því mengi 0 til 15 í skyndiminninu. Önnur línan kemur þar á eftir: stök **a[1][0]**, **a[1][2]**, **a[1][2]** og **a[1][3]** lenda í mengi 16, næstu 4 stök í mengi 17, o.s.frv. Fyrstu tvær línur fylkisins fylla því allt skyndiminnið.

Stök **a[2][0]**, **a[2][1]**, **a[2][2]** og **a[2][3]** lenda svo í mengi 0. Þau slást þá við fyrstu 4 stökin í fyrstu línu fylkisins um það pláss í skyndiminninu. Stökin í línum 0 og 2 í fylkinu a varpast í mengi 0 til 15, en stökin í línu 1 varpast í mengi 16 til 31 í skyndiminninu.



a[0][i] og **a[2][i]**, fyrir **i** = 0 til 64 gefa skelli í hverri einustu ítrun. En stökin **a[1][i]** gefa skelli í fjórðu hverri ítrun, því við náum í 4 stök í hvert sinn og þeim er ekki hent út.
Meðalfjöldi skella per ítrun er því 1 + 0.25 + 1 eða 2.25.
d. Ef skyndiminnið stækkar í 1024 bæti, línustærð áfram 16 bæti og ennþá bein vörpun. Þá fjölgar mengjunum upp í 64 (því 1024 = S*1*16, svo S = 64). Þá fara stökin í fylkislínu 0 (þ.e. a[0][i]) í mengi 0 til 15, fylkislína 1 (a[1][i]) fer í mengi 16 til 31 og fylkislína 2 (a[2][i]) fer í mengi 32 til 47. Þetta þýðir að það verða engir árekstrar á milli línanna í fylkinu og meðalskellafjöldi per ítrun verður þá 0.25 + 0.25 + 0.25 = 0.75.

1. [Próf 2021] Í þessu dæmi höfum við 6-bita orðið **010011**. Það er hægt að túlka það á þrjá vegu sem gildi: *i*) 6-bita heiltala án formerkis (*unsigned*), *ii*) 6-bita tvíandhverfu heiltala (*signed*) og *iii*) 6-bita fleytitala (*floating point*) með 1 formerkisbita, 3 veldisbita og 2 brotbita.

a) Túlkið orðið **010011** sem gildi á þessa þrjá vegu. Sýnið útreikning (sérstaklega í fleytitölunni).
b) Þið megið breyta einum bita í orðinu að ofan og viljið fá sem hæst gildi (þ.e. sem næst +∞). Sýnið og rökstyðjið í hverju tilviki hvaða bita þið mynduð breyta til að hámarka gildið. Það geta verið ólíkir bitar í hverri túlkun.
c) Hvaða einum bita ætti að breyta til að lágmarka gildið (þ.e. komast sem næst −∞) í hverri túlkun? Rökstyðjið hvert tilvik.

hliðrun: 4−3 = 1. Brothlutinn er 11, svo gildið er $1.11*2^1 = 11.12 = 3.5$.

i. Hér er best að breyta efsta bitanum, því hækkar gildið mest. Fáum þá 6- bita orðið **110011** með gildið 32+16+2+1 = 51.

ii. Viljum ekki breyta efsta bita því þá verður talan að mínustölu. Breytum því efsta 0-bita og fáum 6-bita orðið **011011** með gildið 16+8+2+1 = 27.

iii. Viljum hér breyta veldishluta, ef það er hægt, því hann hefur mest áhrif. Auk þess er brothlutinn eins stór og hann getur orðið. Fáum þá **0 110 11**. Nú er veldishlutinn $110_2 = 6$, svo veldið er 6−3 = 3 og gildið á tölunni er $1.11*2^3 = 1110_2 = 14.0$.

i. Best að breyta efsta 1-bita í 0. Fáum þá **000011** með gildið 2+1 = 3.

ii. Best að setja formerkisbitann til að fá mínustölu. Fáum þá **110011** með gildið −32+16+2+1 = −13.

iii. Hér er líka best að setja formerkisbitann. Þá fáum við sama gildi og í a-lið, nema sem mínustölu: −3.5.

2. Segjum að gistið **%rbx** innihaldi **0x800** og gistið **%rdx** innihaldi **0xA**. Hér fyrir neðan eru minnistilvísanir. Í hverju tilviki reiknið út raunverulegt vistfang og rökstyðjið það.
   a. **(%rbx,%rdx)**
   b. **$0x24(%rbx,%rdx,2)**
   c. **$20(,%rbx,8)**
   d. **$-16(%rdx,%rdx,4)**

2. Höfum að gistið **%rbx** inniheldur **0x800** og %rdx inniheldur **0xA**.

(%rbx,%rdx) gefur**0x800+0xA=0x80A**

$0x24(%rbx,%rdx,2) gefur **0x24** + **0x800** + **0xA*2** = **0x0838**

$20(,%rbx,8) gefur**0x14+0x800*8=0x4014**

$-16(%rdx,%rdx,4) gefur**0xA+0xA*4−0x10=0x32−0x10=0x22**

3. Hér fyrir neðan er smalamálsútgáfa af fallinu **long aogb(long a, long b)** sem reiknar einfalda segð (*expression*) út frá viðföngunum **a** og **b**. Athugið að viðfangið **a** kemur í gistið **%rdi**, **b** kemur í **%rsi** og skilagildi fallsins fer í gistið **%rax**.

```
aogb:
    leaq    (%rsi,%rsi,4), %rax
    leaq    (%rdi,%rax,8), %rdx
    leaq    0(,%rdx,8), %rax
    subq    %rdx, %rax
    ret
```

a. Setjið athugasemd fyrir aftan hverja skipun þar sem tilgangur hennar í útreikningnum er útskýrður.
b. Sýnið C-útgáfu af fallinu (hægt að skrifa það með einni **return**-skipun).

3. Gefin smalamálsútgáfa af fallinu **long aogb(long a, long b)**:

a. Athugasemdir fyrir aftan allar skipanir:

```
aogb:
    leaq    (%rsi,%rsi,4), %rax         # rax = 5*b
    leaq    (%rdi,%rax,8), %rdx         # rdx = a + 8*rax
    leaq    0(,%rdx,8), %rax            # rax = 8*rdx
    subq    %rdx, %rax                  # rax = rax - rdx
    ret
```

b. C-útgáfa af fallinu:

Ef við fylgjum smalamálskóðanum, þá er í fyrstu tveimur línunum verið að reikna **(a + 40*b)**. Síðan er það gildi margfaldað með 8 og dregið frá útreiknaða gildinu. Þetta er því fallið:

```
long int aogb( long a, long b  ) {
    return 8*(a+40*b)-(a+40*b);
}
```

Hægt að einfalda (með algebru!) niður í:

```
long int aogb( long a, long b  ) {
    return 7*a + 280*b;
}
```

i. movl %edx, (,%rbx,4)

Lögleg. 4-bæta orðið í **%edx** er sett í minnishólfið með vistfang **Reg[%rbx]*4**.

ii. movzbq $-2, %r8

Lögleg. Bætið með gildið -2 (**0xFE**) er sett í neðsta bætið á gistinu **%r8** og efri hluti gistisins er núllaður. **%r8** inniheldur þá **0x00000000000000FE**.

iii. movswq %ax, %rax

Lögleg. Neðri tvö bætin í **%rax** eru flutt í **%rax** (breytist ekkert!), en síðan er notuð formerkisvíkkun á efri 6 bætin í **%rax**. Þau verða því annað hvort öll 0- bitar, eða öll 1-bitar.

iv. movslw (%rsp), %dx

Ólögleg. Getum ekki víkkað stærra gildi (þ.e. 4 bæti) yfir í minna gildi (2 bæti)!

1. Hér fyrir neðan er smalamálskóði fallsins **long hoho(long x)**. Útskýrið hverja smalamálsskipun og sýnið C útgáfu af fallinu.

```
hoho:
        leaq    31(%rdi), %rax
        testq   %rdi, %rdi
        cmovns  %rdi, %rax
        sarq    $5, %rax
        ret
```

Kóðinn inniheldur skilyrta gagnaflutningskipun (**cmov**), en við getum útfært það í C

með **if**-setningu. Við getum þýtt hverja smalamálsskipun fyrir sig yfir í C kóða:

```
leaq    31(%dri), %rax      long ut = x + 31;
testq   %rdi, %rdi          if ( x >= 0 ) ut = x;
cmovns  %rdi, %rax
sarq    $5, %rax            ut = ut >> 5;
ret                         return ut;
```

Skilyrta gagnaflutningskipunin (**cmovns %rdi, %rax**) setur **x (%rdi)** yfir í **ut (%rax)** ef **x (%rdi)** er ekki neikvæð (**ns**). Aðrar skipanir eru nokkuð augljósar. Fallið verður þá:

```
long hoho (long x) {
    long ut = x + 31;
    if ( x >= 0 ) ut = x;
    ut = ut >> 5;
    return ut;
}
```

Athugið að síðasta skipunin hliðrar **ut** um 5 sæti til hægri, þ.e. deilir með 32 (=2^5) upp í **ut**. Til að deilingin rúnnist rétt fyrir neikvæðar tölur þarf að bæta við hliðrun ef talan er neikvæð (en ekki ef hún er jákvæð). Það er tilgangurinn með fyrstu 3 smalamálsskipununum. Það væri því hægt að skrifa fallið á einfaldari hátt og fá sömu smalamálsþýðingu:

```
long hoho (long x) {
    return x / 32;
}
```

2. Hér fyrir neðan en smalamálskóði fallsins **long whi(long k)**. Fallið inniheldur eina **while**-lykkju. Skrifið fallið í C og útskýrið hvernig einstakið hlutar C fallsins samsvara smalamálsskipununum hér fyrir neðan.

```
whi:
        movl    $0, %eax
        jmp     .L2
.L3:
        leaq    -1(%rdi), %rdx
        andq    %rdx, %rdi
        xorq    $1, %rax
.L2
        testq   %rdi, %rdi
        jne     .L3
        ret
```

2. Gefinn smalamálskóði fallsins **long whi(long k)**. Líka gefið að fallið hafa **while**-lykku.

Smalamálskóðinn er skipulagður eins og **while**-lykkja af gerð 1. Lykkjan heldur áfram á meðan **k (%rdi)** er ekki 0.

Innihald lykkjunnar er 3 skipanir. Fyrst er 1 dreginn frá **k** og útkoman sett í **%rdx**. Síðan eru **%rdx** og **k** OG-uð saman. Þetta er C kóðinn " **k=k&(k-1)**". Loks er 1 XOR-aður við gistið **%rax**. Sú aðgerð breytir neðsta bitanum (*flips*) í **%rax**. Í upphafier **%rax** sett sem 0. Síðan verður neðsti biti **%rax** 1, í næstu ítrun verður hann 0 aftur, o.s.frv.

```
long whi(long k) {
        long p = 0;
        while (k != 0) {
            k = k & (k-1);
            p = p ^ 1;
        }
        return p;
    }
```

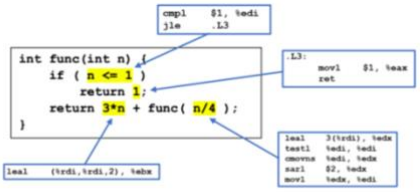
Þessi kóði notar bitatrikk til að eyða út einum 1-bita í **k** í hverri ítrun. Skilagildi fallsins er 0 ef fjöldi 1-bita er jöfn tala, en 1 ef það er oddatala. Fallið reiknar því út svokallaðan varbita (*parity bit*) fyrir bitastrenginn í **k**. *[Þið þurftuð ekki að finna þetta út, bara til fróðleiks!].*

3. [Próf '21] Hér fyrir neðan er x86-64 smalamálsútgáfa af endurkvæma fallinu **`func`**

```
func:
        cmpl    $1, %edi
        jle     .L3
        pushq   %rbx
        leal    (%rdi,%rdi,2), %ebx
        leal    3(%rdi), %edx
        testl   %edi, %edi
        cmovns  %edi, %edx
        sarl    $2, %edx
        movl    %edx, %edi
        call    func
        addl    %ebx, %eax
        popq    %rbx
        ret
.L3:
        movl    $1, %eax
        ret
```

a. Skrifið jafngilda C útgáfu af þessu falli. Til að hjálpa ykkur við það er hér fyrir neðan beinagrind af fallinu sem þið getið fyllt inn í. Rökstyðjið sérstaklega hvaða smalamálsskipanir standa á bakvið þann kóða sem þið setjið inn

b. Teiknið upp hlaðaramma (stack frame) fyrir fallið. Sýnið stöðuna þegar þrjú endurkvæm köll hafa orðið. Tilgreinið einstaka hluta hvers hlaðaramma



Skilyrðið í **`if`**-setningunni fæst vegna þess að fyrstu tvær skipanirnar í fallinu (**`cmpl $1, %edi`** og **`jle .L3`**) hoppa yfir í **`.L3`** ef **`n`** er ≤ 1.

Ef **`n`** ≤ 1 þá er hoppað í **`.L3`** og þar er sett 1 sem skilagildi og þess vegna er fyrri **`return`**-skipunin með 1.

Áður er kallað er endurkvæmt á fallið þá er kóði sem er dæmigerður þegar verið er að deila með heilu veldi af 2. Þá er notuð hliðrun um 2 sæti til hægri (**`sarl $2, %edx`**), en áður þarf að bæta við hliðrun ef talan neikvæð, til þess að rúnnunin verði rétt.

Að lokum er **`n`** (**`%rdi`**) margfaldað með 3 (reyndar gert efst í kóðanum) og það lagt við útkomuna úr endurkvæma fallinu.

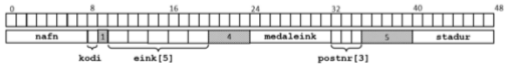
1. Hér fyrir neðan er skilgreining á færslu (*struct*) í C forriti:

```
struct upplys {
    char *nafn;
    char kodi;
    short int eink[5];
    double medaleink;
    char postnr[3];
    char *stadur;
};
```

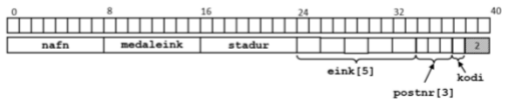
a. Sýnið teikningu af þessari færslu sem tekur tillit til uppröðunarkröfu (alignment requirements) x86-64. Hversu mörg bæti tekur þessi færsla?

b. Er hægt að umraða sviðum (fields) færslunnar þannig að hún taki minna minnispláss? Sýnið teikningu af nýrri uppröðun, ef það er hægt

a. Stærsta uppröðunarkrafan er 8 bæti, því stærsta grunntagið er 8 bæti, sem eru tveir bendar (nafn og stadur) og ein double-breyta (medaleink). Hér fyrir neðan er mynd af færslunni eins og hún er uppsett. Hún tekur 48 bæti og það eru 10 bæti sem eru fylling (= 1 4 + 5). Það ætti því að vera hægt að gera betur!



b. Með því að umraða sviðunum náum við að minnka stærðina niður í 40 bæti. Ekki er hægt að fara neðar, því uppröðunarkrafan er 8, svo lengdin verður að vera heilt margfeldi af 8. Samtals eru öll svið in 38 bæti, svo 40 er minnsta heila margfeldi af 8 sem hægt er að nota.



`main`-skrá:

Tákn	Sterkt/veikt	Skýring
`fall`	veikt	skilgreining
`b`	sterkt	upphafsstillt breyta
`c`	veikt	óupphafsstillt breyta
`d`	veikt	óupphafsstillt breyta
`main`	sterkt	skilgr. á falli
`printf`	veikt	notkun/skilgr. á falli

`fall`-skrá:

Tákn	Sterkt/veikt	Skýring
`a`	veikt	óupphafsstillt breyta
`b`	veikt	óupphafsstillt breyta
`c`	sterkt	upphafsstillt breyta
`d`	sterkt	upphafsstillt breyta
`fall`	sterkt	skilgr. á falli

c. Úttak forritsins er

```
a: 1, b:7, c:3, d:5
```

Útskýring

- a er static breyta í main skránni, víðvær innan skrárinnar og hefur gildið 1 þar, a breytan í fall.c er önnur víðvær breyta en það er ekki vísað í hana úr main.c
- b er víðvær breyta sem er skilgreind í main.c, fær gildið 2 í upphafi keyrslu, hún er skírgreind sem extern int í fall.c og er breytt í 7 við kallið á fall()
- c er skilgreind í fall.c og vísað í hana í main.c, en inni í main fallinu er staðvær breyta c sem skyggir á víðværu breytuna og sú fær gildið 3 og er prentuð út
- d er víðvær breyta, skilgreind í fall.c, main.c hefur skírgreiningu með extern int sem er sama breyta, d er upphafsstillt sem 5 í fall.c. Í fallinu fall() er skilgreind static (ein breyta fyrir öll fallaköll á fall()) staðvær breyta með gildissvið inni í fall(), línan d=9 breytir static breytinni en ekki víðværu breytunni. Inn í main prentast því 5 út.

2. [Próf '21] Tölva nokkur hefur 16 bita sýndarvistföng og 10 bita raunvistföng. Síðustærðin er 16 bæti. Tölvan hefur 2-vítt TLB með samtals 8 sætum.

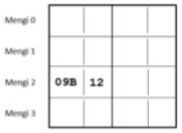
a. Rissið upp skiptingu sýndar- og raunvistfanga í VPN, VPO, PPN, PPO, TLBT, TLBI og sýnið fjölda bita í hverjum hluta.
b. Teiknið upp TLB sem inniheldur gildi fyrir sýndarvistfangið **`0x26E7`**. Setjið rétt merki (*tag*) í löglegt sæti og búið til eitthvert gildi af réttri stærð fyrir PPN-gildið. Það þarf ekkert að vera í hinum sætunum.
c. Hvers konar sýndarvistföng varpast í mengi númer 1 í þessu TLB? Hve mörg eru þau? Rökstyðjið.
d. Útskýrið hvort það gæti gerst að sama sýndarvistfangið, t.d. vistfangið í b-lið að ofan, varpist í tvö ólík raunvistföng á ólíkum tímum í keyrslu forrits. Rökstyðjið svar ykkar í nokkrum orðum.

2. Gefin er tölva með 16-bita sýndarvistföng og 10-bita raunvistföng. Síðustærð er 16 bæti. Tölvan hefur 2-vítt TLB með samtals 8 sætum, sem þýðir að fjöldi mengja er 4 (= 8/2).

a. Sýndarvistföngin skiptast í 12-bita VPN og 4-bita VPO (því síðustærð er 2^4 = 16 bæti). Þar sem TLB hefur 4 mengi þá er TLBI 2 bitar og TLBT er þá 10 bitar (= 12 − 2).

Raunvistföngin skiptast í 6-bita PPN og 4-bita PPO (sama og VPO).

b. Sýna TLB með gildi fyrir sýndarvistfangið **`0x26E7`**. Gildið á VPN er þá **`0x26E`**, eða 0010 0110 1110 í bitum. Neðstu 2 bitarnir mynda TLBI, svo mendisnúmerið er 2 (= 10_2). Efri 10 bitarnir mynda TLBT, sem er **`0x09B`**. PPN gildið er 6 bitar, svo við getum t.d. notað gildið **`0x12`** fyrir það. Hér fyrir neðan er gróf mynd af TLB, með 4 mengjum og plássi fyrir tvö síðutöflustök í hverju mengi.



c. Hvers konar sýndarvistföng varpast í mengi númer 1 í TLB. Þá eru bitarnir tveir í TLBI með gildið 01, en aðrir bitar geta verið hvað sem er. Þetta eru þá vistföng af gerðinni **`xxxxxxxxxx01xxxx`**. Það eru þá 14 bitar sem geta verið hvað sem er, svo það eru 2^{14} möguleg gildi á þeim.
d. Það getur gerst að sama sýndarvistfangið varpist í tvö ólík raunvistföng í sömu keyrslu forrit. Til dæmis þegar síðan kemur inn í minnið í fyrsta sinn þá er hún sett á tiltekinn stað í aðalminninu. Svo getur henni verið hent út fyrir einhverja aðra síðu. Þegar síðan kemur svo inn aftur í aðalminnið þá lendir hún ekki endilega á sama stað í aðalminninu og hún var áður og því eru raunvistföngin önnur.

4. [Próf '21, lítillega breytt] Hér fyrir neðan eru tvær C forritaskrár sem innihalda víðværar breytur.

```
#include <stdio.h>
void fall();

static int a = 1;
int b = 2;
extern int c;
extern int d;

int main() {
   int c = 3;

   fall();
   printf("a:%d, b:%d\n", a, b);
   printf("c:%d, d:%d\n", c, d);
   return 0;
}
```

```
int a;
extern int b;
int c = 4;
int d = 5;

void fall() {
    static int d;

    a = 6;
    b = 7;
    c = 8;
    d = 9;
}
```

a. Farið í gegnum báðar skrárnar og fyrir hverja skilgreinda breytu gefið upp hvers konar breytu er um að ræða og hvar breytan sé vistuð í minnisrými ferlisins (þ.e. hlaði, kös, **`.data`**-svæði, ...).

b. Farið í gegnum báðar skrárnar og fyrir hvert víðvært tákn í skránni segið hvort það sé sterkt eða veikt tákn.

c. Hvert er úttak forritsins þegar það er keyrt? Rökstyðjið hvert gildi fyrir sig.

4. Breytur í main og fall skrám

a. Allar breytur

`main`-skrá:

Breyta	Staðsetning	Skýring
`static int a = 1;`	í **`.data`** svæði	Upphafsstillt *static* víðvær breyta
`int b = 2;`	í **`.data`** svæði	Upphafsstillt víðvær breyta
`extern int c;`	í **`.bss`**/**`.data`** svæði	Óupphafsstillt hér
`extern int d;`	í **`.bss`**/**`.data`** svæði	Óupphafsstillt hér
`int c = 3;`	á hlaða	Staðvær breyta

`fall`-skrá:

Breyta	Staðsetning	Skýring
`int a;`	í **`.bss`** svæði	Óupphafsstillt víðvær breyta
`extern int b;`	í **`.bss`**/**`.data`** svæði	Óupphafsstillt hér
`int c = 4;`	í **`.data`** svæði	Upphafsstillt víðvær breyta
`int d = 5;`	í **`.data`** svæði	Upphafsstillt víðvær breyta
`static int d;`	í **`.bss`** svæði	Óupphafsstillt *static* staðvær breyta

b. Sterk og veik tákn í skránum

2. Hér til hliðar er fallið **`para()`** sem býr til ný ferli og prentar út gildi á breytunni **`x`**.

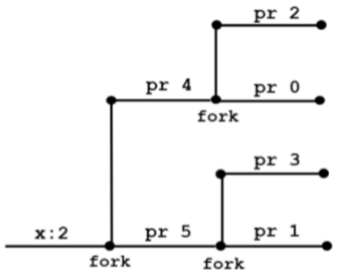
a. Teiknið ferlarit (*process graph*) fyrir fallið og merkjið inn á það gildi breytunnar **`x`** á einstökum stöðum.

b. Sýnið þrjár mögulegar raðanir á úttaksgildunum þegar forritið er keyrt.

c. Er mögulegt að úttaksgildin komi í strangt lækkandi röð? Rökstyðjið.

```
void para() {
   int x = 2;
   if (fork())
      x += 3;
   else
      x += 2
   printf("%d ", x);
   fflush(stdout);
   if (fork())
      x -= 4;
   else
      x -= 2;
   printf("%d ", x);
   fflush(stdout);
   exit(0);
}
```

a. Ferlarit



b. Þrjár mögulegar raðanir á úttakinu:
513402
402531
541302
c. Já, röðin 543210 getur komið fyrir

3. Sjálfstæðar spurningar um ferli:

a. Gefið dæmi um frábrigði (*exception*) þar sem eðlilegt er að framkvæma aftur sömu skipun þegar komið er til baka úr frábrigðinu.
b. Gefið dæmi um frábrigði (*exception*) þar sem eðlilegt er að hætta keyrslu ferlisins (*abort process*).
c. Þegar stýrikerfið framkvæmir samhengisskipti (*context-switching*) frá ferli P_1 yfir til ferlis *P2*, hverju af eftirfarandi þarf að forða (*save*) þegar ferlinu P_1 er skipt út: *i*) gistu, *ii*) innihald L1 skyndiminnis, *iii*) innihald TLB? Útskýrið í nokkrum orðum svar ykkar í hverju tilviki.
d. Þegar nýtt ferli er búið til með **`fork()`** fallinu hvaða eiginleika foreldraferlisins afritar stýrikerfið til þess að búa til afkomendaferlið?

i. Dæmi um frábrigði þar sem eðlilegt er að framkvæma aftur sömu skipun þegar komið er til baka úr frábrigðinu er síðutöf (page fault). Þá er síðan komin í minni og þá getur skipunin sem olli síðutöfinni loks náð í innihald minnishólfsins.
ii. Dæmi um frábrigði þar sem eðlilegt er að hætta keyrslu er heiltöludeiling með núlli. Önnur dæmi eru vélbúnaðarvillur, t.d. í minni.
iii. Við samhengisskipti þarf að forða gistum, því þau eru hluti af stöðu ferlisins á hverjum tíma. Það þarf ekki að forða innihaldi L1 skyndiminnisins, því það endurnýjast bara smátt og smátt þegar nýja ferlið hefur keyrslu. Síðutafla hvers ferlis er sjálfstæð og því mun innihald TLB ekki nýtast nýja ferlinu, en líklega er skynsamlegast að forða því ekki og láta það bara endurnýjast eins og innihald L1 skyndiminnisins.
i. Þegar nýtt ferli er búið til þá afritar stýrikerfið allt sýndarminnissvæði foreldrisins (þar með talið forritssvæði, kös, hlaða og sameiginleg forritasöfn), einnig handföng á opnum skrám. Afkomendaferlið er eins og foreldraferlið, nema það hefur annað ferlanúmer (*pid*).

1. Hér til hliðar er forritsbútur sem prentar út streng sem er samsettur úr **`0`** og **`1`**.

a. Hversu mörg 0-tákn munu prentast út og hversu mörg 1-tákn?
b. Hve mörg af hverju tákni munu prentast út ef við setjum breytuna **`k`** í stað fastans 3 í **`for`**-lykkjunni? Rökstyðjið svarið.

```
int i;
for (i=0; i<3; i++) {
    printf("0");
    fflush(stdout);
    fork();
}
printf("1");
exit(0);
```

a. Fjöldi 0-tákna og fjöldi 1-tákna: for-lykkjan hefur þrjár ítranir. Í upphafi fyrstu ítrunar er aðeins eitt ferli í gangi og það prentar úr eitt 0-tákn. Í upphafi næstu ítrunar eru komin tvö ferli í gang og þau prenta bæði út eitt 0-tákn. Í upphafi þriðju ítrunar eru fjögur ferli komin í gang og hvert þeirra prentar út eitt 0-tákn. Samtals eru þetta 1+2+4 = 7, 0-tákn. Í lok þriðju ítrunar eru búin til fjögur ný ferli og eftir for-lykkjuna eru því átta ferli í gangi. Hvert þeirra prentar út eitt 1-tákn, samtals 8 1-tákn.
b. Ef við höfum **`k`** ítranir í stað þriggja, þá verður fjöldi 0-tákna $1+2+...+2^{k-1}$, eða samtals 2^k-1. Fjöldi 1-tákna verður þá 2^k. Þetta sést vel á ferlaritinu fyrir forritið þegar **`k`** = 3: