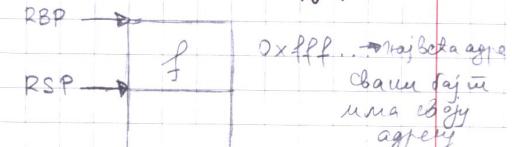
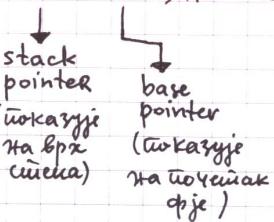
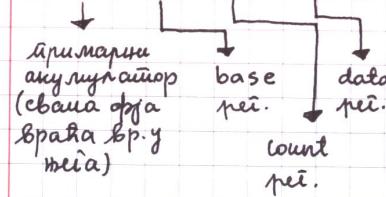


• 16 64-битних регистара :

RAX, RBX, RCX, RDX, RSI, RDI, RSP, RBP, R8-R15

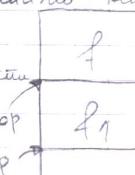


\* ако кадамо call f1

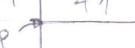
f1 : *напушта бр. Rbp-а на стеку, па следом долази да испуни уз аргументе*

= enter 0,0

*push Rbp*  
*mov Rbp, Rsp* → аргумент  
рсп



*mov Rsp, Rbp*  
*pop Rbp* → *извлачи бр. са стека и свише је вредност ресултат*



leave  
ret

\* add op1, op2 → op1 = op1 + op2

mov rax, 45250 *операција као 64bit јер је rax 64bit*

али ако је број величине 1, онда ће бити 32bit

бит 1111...1111 и онда свици су 64bit

величине, ако су због 32 битне највеће 32 бита, тј. 0000 0000 1111...1111

и то би било 4 бити -1. Неки број величине број

1. Начинати Hello world!

.intel\_syntax noprefix

.data → секција со иницијализовани подацима

fmt: .asciz "Hello world!\n"

.text → секција со кодом

.global main

main:

enter 0,0

#printf(fmt) ≡ f(rdi)

Задача је да се користи rdi уместо rsi  
агрејв на којој се налази fmt

Не може да се користи rdi, fmt је задато аргумент  
ог fmt да се користи rdi

lea rdi, fmt → аргумент је користи у rdi  
call printf  
leave  
ret

\*агрејв увек користимо у чео регистар јер су оне 64 битне

2. .data

fmt: .asciz "%ld\n"

.text

.global main

main:

enter 0,0 / xor rsi,rsi → rsi=0

lea rdi, fmt

mov esi, -1

→ esi = 1111 1111

call printf

leave

ret

64bit  
је је 64 битно %ld  
унесе 32 бита  
унесују се 0000  
које је само -1  
сместава у 32 битна  
реч.

изједначава се иницијализација 64 бит  
изједначава је rsi, је esi је 0000 1111  
а то је 0000 0000 1111 1111 ≠ -1

• Регистри који трансформују позивачките функције су  
rbx, rbp, r12-15 → сагријав обикновени регистри  
мора сачувати (на чину)  
ако се меша у некој обје

• Регистри који трансформују познатој обје су  
rax, rdi, rsi, rdx, rcx, r8-11  
(врх стека поизваније обје мора да биде поравнат са  
агрејвом десктолом као 16

↓  
може да узрокује segmentation fault  
ако нисе поравнати )  
извршена вредност се кува у rax

3. Како пагу мов

.data

fmt: .asciz "%ld\n"

x: .int 0,1

text

.global main

main:

enter 0,0

xor rsi,rsi

lea rdi,fmt

mov rsi,x

call printf

leave

ret

rdi rsi  
#printf(fmt, x)

упакуј 0,0, се смести на  
нулих 32 бита а глајн дрој на високих  
32  
мисајујују је 0000 1111 1111 1111  
смешава се 0000 0000 1111 1111  
изједначава је rsi и rdi и  
изједначава је rsi и rdi и

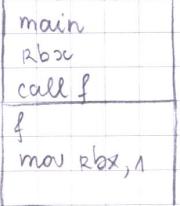
## II negera

f(rdi, r8, rdx, rcx, r8, r9)

рекурсији који  
тумицагају извештај ојбу  
(извештаја бп. то басе ојб  
се сместава у rax)

f(rbae, rbp, R12 - R15)

рекурсији који  
тумицагају извештај ојбу



→ не може → push rbx  
је main, mov rbx, 1  
која је извештаја, pop rbx  
којим је  
ујачији f нура  
извештаја се још ојбу  
је rax на чинију, да ојгс  
нуше га реди (2 буји  
нуна љубије)  
на крају нура ојчији  
је вранији извештаји  
блоготије је rbx

1. Највијам ојбу која садира елементе у нури

.intel-syntax noprefix

.data

fmt: .asciz "%d\n"

array: .int 5, 10, 15, 20, 25

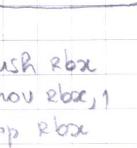
n: .int 5

.text

global main

main:

enter 0,0



# sum (array, n) → бака здуп сместет је rax (eax)  
rdi rsi → којишћићемо јеј смо наћемо  
га је n int виј. 32 bit

lea rdi, array → јеј је array то вако ћемо, виј  
mov esi, n → ако је  
call sum

→ здуп интоба је int → 32 bit

#printf (fmt, cima koја је у eax)

rdi rsi (esi)

lea rdi, fmt → код сумом га је изменета неј.  
mov esi, eax → rax који тумицага ојбу који  
call printf → смо извештаји, нура њој га  
"оријентиса" rax на крају

xor Rax, Rax  
leave  
ret

sum:

enter 0,0

xor eax, eax → sum = 0

xor ecx, ecx → i = 0

loop\_begin:

→ while i != n  
ecx esi

cmp ecx, esi

je loop\_end → ако јеј је грав (i=n) угуји на  
крај.

addl eax, [rdi + 4 \* rcx]



5 10 ...

↑ ~

array (rdi)

4B = 32b

сумираји  
нума дигар

[B + S \* I]

↓

Бака је грав елементија

inc ecx  
jmp loop\_begin

loop\_end:  
leave  
ret

→ return res = eax

2. за јеа јеста држа иницијалне стапе, пасиви, конкретни и иницијални.

.intel-syntax noprefix

.data

readFmt: .asciz "/d%d"  
writeFmt: .asciz "%d\n"  
writeFmt1: .asciz "%ld (%d)\n"  
.bss → иницијализоване стапе  
.lcomm x, 4 → x има 4 байта  
.lcomm y, 4

.text

.global main

main:

enter 00

#иницијално очејаваје , scanf (readFmt, &x &y)  
lea rdi, readFmt

lea rsi, x

lea edx, y

call scanf

apreca

rsi

rdi rsi edx

#зуп увеју  
lea rdi, writeFmt

mov esi, x

add esi, y

call printf

printf (writeFmt, x+y)  
rsi (esi)

#према увеју  
lea rdi, writeFmt

mov esi, x

sub esi, y

call printf

# иницијални  
lea rdi, writeFmt

imul op1, op2 (измене сајају дјелују)

imul op

ano je op 32bit брежњак тегните eax \* op  
се смешавају edx : eax

ano je op 64bit пејс се смешавају  
edx : rax

mov eax, x

imul dword ptr y → осигурава жеје је 32bit

mov esi, eax

→ иницијално иницијално у eax као y

call printf

#конкетек и осимак

idiv op

ano je op 32bit : edx : eax

осимак конкретик

ano je op 64bit : rdx : rax

mov eax, x

cdq → иницијално (измене) eax на edx : eax

idiv dword ptr y → жеје edx : eax као 32bit y

lea rdi, writeFmt1

cdqe → y eax je конкретик и хвено гаја  
иницијално као long long int, нукало га  
измене иницијално еакс на раг

mov rsi, rax

call printf

xor rax, rax

leave

ret

4. учи се елементи, димензија маса и реаз, иницијализација је ако се елементи напади у маси, масе 'n'. Користимо писањем да пишемо.

.data

read: .asciz "y d"  
write: .asciz "x c\n"

.bss

.lcomm value, 4

.text

main:

enter 0,0

{ #иницијализација елементи и чврсто је Rbx  
lea rdi, read  
lea rsi, value  
call scanf

push rbx → rbx је рејт поизважујући објекат који чврсто  
push rbx на чврсто

иницијализација 2 пута на стек  
запосио стек нова го бидеје  
поглавнице на 16 байтова пре  
постави свеје где

mov ebx, value

{ #иницијализација думичну масу и спомешамо је у ecx  
lea rdi, read  
lea rsi, value  
call scanf

mov ecx, value

#проверава да ли је думична маса већа од 0  
jecxz not-found → jump if ecx = zero

next\_element:

#иницијализација 1 по 1 елементи маса и проредило са  
иницијализацијом

# други елементи који је прескаче да се у чврсто  
се у неизвршењу труба на чврсто је да објекат  
scanf утицајем ће је супријеј

push rcx } је да спољу scanf нова је  
push rcx проредило вредност у Rcx  
a нова 2 пута

{ lea rdi, read  
lea rsi, value  
call scanf

pop rcx  
pop rcx

cmp ebx, value → инверсија = добијати ?  
je done

dec rcx → чувајују елементи да  
иницијализирају

cmpl rcx, 0  
je not-found

jmp next\_element

done:  
mov rsi, 'y'  
jmp print

jmp print

not\_found:  
mov rsi, 'n'

print:

lea rdi, write  
call printf

xor rcx, rcx  
pop rbx  
pop rbx

Loop читајује ecx и проверава  
да ли је ecx нула  
нује

loopne читају како претходно  
+ ZF (zero flag) је постовала

loopne читају како loopne  
само читају захтева да  
ZF нује постовављен

### III Hesjeva

1.  
main.c :

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

int secret(int x);

int main()
{
    int broj, tajni_broj;
    scanf("%d", &broj);
    assert(broj >= 0);
    tajni_broj = secret(broj);
    printf("%d\n", tajni_broj);

    return 0;
}
```

secret.s:

ty rdi ce hanasu ymnuatu dnoj  
#rax u rdx za gresive

.intel-syntax noprefix  
.data

.text

.global secret

secret :

```
    enter 0,0
    push rbx
    push rcx
```

```
mov rcx, 1
xor R8, R8
```

```
mov r9, 10
```

#usgbajano jeguy yuephy  
geniusu ebanu ymca 10 godicau  
godicu cnegety yuephy

rdi = 8169 kap.

```
mov rdx, 0
```

```
mov Rax, rdi
```

```
div R9
```

cnybajano

kon.yuephu

y rde

urbx

```
cmp rdi, 0
```

je done

rbx

= 9

rdi

= rax

= 816

outaiaak

komutak

{ rax = 8169 : 10 = 816

rdx = 9

} vena bime yuepha  
za usgbajane

next:

#usgbajano cnegety yuephy

```
mov rdx, 0
```

```
mov rax, rdi
```

```
div r9
```

rax = 816

rax = 81, rdx = 6

```
add rbx, rdx
```

rbx = rbx + rdx = 15

```
mov rdi, rax
```

" " " "

rdi = 81

cmp rbx, 10

ano je rbx < 10 ujemogane

jb sredjeno

push RAX = 81

push RDI = 81

push R8 = 0

push RDX = 6

cnybajano bp. reiznara

ta cmeuy jep hemo ux

y hanabu y mohga

brimnem je ujemao

JI y = 1 ujemao istezem  
mnozitvem

1. krapay

sredi:

mov rdi, rbx  
call secret

сврјај уважје је  $rbx \geq 10$   
 $rdi = rbx = 15$   
изведене стји за обај дрој 15

cmp rax, 10  
ja sredi

ауо је извршена др. дрој  
изведене > 10

mov rbx, rax

незваник rax = 6  
 $rbx = 6$

pop rdx  
pop r8  
pop rdi  
pop rax

sredjeno:

#изјавио дрој у свемиринија је r8

push rdx                   $rdx = 1$   
mov rax, rbx             $rax = 6$   
mul rcx                 $rax = rax \times rcx = 6$   
add r8, rax             $r8 = 6$

mov rax, rcx             $rax = 1$   
mul r9                 $rax = 1 \times 10 = 10$   
mov rcx, rax             $rcx = 10$

pop rdx

mov rbx, rdx             $rbx = 1$

cmp rdi, 0  
jne next

pop rcx  
pop rbx

mov rax, r8             $rax = 6$

cmp rax, 10  
jb done

previo:  
mov rdi, rax  
call secret

done:

leave  
ret

2. main.c:

rdi      rsi  
rsi      rs

extern int count-primes(int a, int b);

int main() {

int a, b, rez;

scanf ("%d%d", &a, &b);  
rez = count-primes(a, b);  
printf ("%d", rez);

return 0;

}

count-primes.s:

count-primes:

enter 0,0

mov r9, rdi     $\rightarrow r9 = a$   
mov r10, r8     $\rightarrow r10 = b$   
xor r8, r8

hyp.  $a = 2$   
 $b = 100$

следећи:

$\rightarrow$  mov rax, r8             $rax = 0$   
cmp r9, r10             $r9 > r10 \rightarrow$  упаж  
ja done

mov rcx, 2

mov rdx, 0

mov rax, r9             $rax = r9$

div rcx                 $rax = rax : rcx = 0 / 2 = 1$

mov r15, rax             $r15 = 1$

jmp is-prime

prost:

add r8, 1

jmp sledeli

is-prime:

mov rax, r9      rax = 2

add r9, 1      r9 = 3

loop: {  
    cmp r9, 2  
    r9 == 2  
    je sledeli

    mov rcx, 2

prime-petyica:

cmp rcx, r15      2 > 1

ja prost

mov rdx, 0

mov rax, r9      r9 = 4

sub rcx, 1      rax = 4

div rcx

cmp rdx, 0

je sledeli

add rcx, 1

jmp prime-petyica

done:

leave

ret

3.  $f(x, y, z) = \begin{cases} 1 + f(f(x-1, y, z), f(y-1, z, x), f(z-1, x, y)) & \text{if } rdi < rsi & \text{and } rdx < rsi \\ z & \text{otherwise} \end{cases}$

takuchi:

enter 0, 0

cmp rdi, rsi      y >= y & rsi <= x & y <= x

ja poz

mov rax, rdx

jmp done

y >= y & rsi <= x & y <= x

scy u posymirat j i z

poz:

push rdi

push rsi

push rdx

sub rdi, 1      x - 1

call takuchi

mov r8, rax      r8 = uobravta qf qye  
                      wj. f(x-1, y, z)

add rdi, 1      x - 1 + 1 = x

pop rdx      rdx = z

push rdx

push rsi

push rdi

mov r9, rsi      r9 = y

mov rsi, rdx      rsi = rdx = z

mov rdx, rdi

mov rdi, r9

sub rdi, 1

call takuchi

mov r9, rax      r9 = f(y-1, z, x)

pop rdi

pop rsi

pop rdx

mov r10, rdx      r10 = z  
mov rdx, rsi      rdx = y  
mov rsi, rdi      rsi = x  
mov rdi, r10  
sub rdi, 1  
(all taken)  
mov r10, rax      r10 = f(z-1, x, y)

```
mov rdi,r9  
mov r8,r9  
mov rdx,r10  
call tahleny  
add rax,1
```

done:

## IV Неделя

## Раг са реалним пројективим

- 2 типа инструкций:

- SSE - једнострука преносимост (32-битни податак)



31 30 ... 23 дракула (23 дниа)

Броят се формира на основа :  $(-1)^{b_{21}} \times 2^{(b_{30} b_{29} \dots b_{23})_2} \times (1.b_{22} \dots b_{20})_2$

- SSE2 - двострукта тиреунгносці (64 бітів)



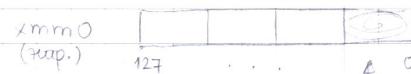
63      52      e      обработка (52 минут)

$$\text{дво} \times \text{дво} = (-1)^{b_{63}} \times 2^{e-1+023} \times (1.b_{51} \dots b_0)_2$$

- 16 128-димензияларда  $x_{min} = x_{max} = 15$

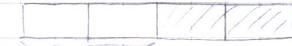
SSE иницијатура ће бити у складу са стварним

movss :



обје споменка, на најниче датуме  
дј. научних 32 дана је  
размјено а 32 дана преје доје

movlps:



составляю кипролексико

mov hps:



Hyponephele

movhlps xmm0, xmm1

xmm0	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
------	----------------	----------------	----------------	----------------

xmm1	b <sub>3</sub>	b <sub>2</sub>	<u>b<sub>1</sub></u>	b <sub>0</sub>
------	----------------	----------------	----------------------	----------------

→ xmm0	<u>b<sub>1</sub></u>	b <sub>0</sub>	a <sub>1</sub>	a <sub>0</sub>
--------	----------------------	----------------	----------------	----------------

movhlps xmm0, xmm1

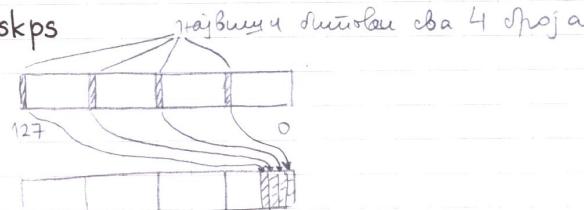
Бити остатку остатку  
неприменяется

xmm0	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
------	----------------	----------------	----------------	----------------

xmm1	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
------	----------------	----------------	----------------	----------------

→ xmm0	a <sub>3</sub>	a <sub>2</sub>	b <sub>3</sub>	b <sub>2</sub>
--------	----------------	----------------	----------------	----------------

movmskps



addps xmm0, xmm1

xmm0	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
------	----------------	----------------	----------------	----------------

xmm1	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
------	----------------	----------------	----------------	----------------

→ xmm0	a <sub>3</sub> +b <sub>3</sub>	a <sub>2</sub> +b <sub>2</sub>	a <sub>1</sub> +b <sub>1</sub>	a <sub>0</sub> +b <sub>0</sub>
--------	--------------------------------	--------------------------------	--------------------------------	--------------------------------

addss xmm0, xmm1

→ xmm0	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
--------	----------------	----------------	----------------	----------------

cmpps xmm0, xmm1, 0 хеккегагы = equal

1 хекк. = less than

2 хекк. = less than or equal

3 хекк. = unordered

4 хекк. = greater than or equal

5 хекк. = greater than

shufps xmm0, xmm1, хекк. или фнг. дпж

нрп. shufps xmm0, xmm1, 93 хекк.

или 10010011 дпж.

xmm0	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
------	----------------	----------------	----------------	----------------

xmm1	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
------	----------------	----------------	----------------	----------------

10 01 00 00

xmm0	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	b <sub>3</sub>
------	----------------	----------------	----------------	----------------

ауд же реальная итог:

shufps xmm0, xmm0, 0b10010011

xmm0	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
------	----------------	----------------	----------------	----------------

→ xmm0	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	a <sub>3</sub>
--------	----------------	----------------	----------------	----------------

unpckhps xmm0, xmm1

xmm0	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>
------	----------------	----------------	----------------	----------------

xmm1	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>
------	----------------	----------------	----------------	----------------

→ xmm0	b <sub>3</sub>	a <sub>3</sub>	b <sub>2</sub>	a <sub>2</sub>
--------	----------------	----------------	----------------	----------------

кординац то 2 тофана  
са бүйүр аялал

unpcklps

xmm0	b <sub>1</sub>	a <sub>1</sub>	b <sub>0</sub>	a <sub>0</sub>
------	----------------	----------------	----------------	----------------

## 1. maksimum niza

main.c :

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
```

```
extern void maximum(int n, float *P, float *rez);
int main() {
```

int n, i;

float \*P, \*rez;

scanf ("%d", &n);

assert (n > 0);

assert ((P = malloc (n \* sizeof (float))) != null);

for (i=0; i < n; i++)

scanf ("%f", &P[i]);

maximum (n, P, rez);

printf ("Maksimum niza je: %f\n", \*rez);

free (P);

return 0;

maximum\_prvi\_korak.s :

maximum:

enter 0,0

mov rg, rdi

→ rg = n

mov r10, rsi

→ r10 = \*P      usmernica niza

mov r8, rdx

**upravlja ga**  
**mu upravlja**  
**upravlja**  
**ob učinak.**  
**ubekce**  
**izmeu**

mov rax, 1

cpuid

test rdx, 0x2000000

jz not-supported

**mauve**  
**cyber**  
**izmeu**

```
{ mov rdx, rsp
  and rsp, 0xfffffffffffff0
  sub rsp, 512
  fxsave [rsp]
```

mov rcx, 1

movss xmm0, [r10]      na gaje max obaj

xmm0 [ ] [ ] [ ]  
 127 [r10]

movss [r8], xmm0

r8 [ ] [ ] ~

**ybek**  
 fxrstor [rsp]  
 mov rsp, rdx

done:

leave

ret

**not supported:**  
**ybek**  
 mov rde, 1  
 call exit

maximum.s :

maximum:

enter 0,0

mov rg, rdi

→ rg = n

mov r10, rsi

→ r10 uokazuje na doširak niza

mov r8, rdx

→ perynatia

{ mov rax, 1
 cpuid
 test rdx, 0x2000000
 jz not-supported

$\left\{ \begin{array}{l} \text{mov rdx, rsp} \\ \text{and rsp, 0xff...f0} \\ \text{sub rsp, 512} \\ \text{fxsave [rsp]} \end{array} \right.$

$\text{mov rcx, 1} \rightarrow \text{rcx = 0 проар}$

$\text{movss xmm0, [r10]} \rightarrow \text{xmm0 } \boxed{\quad \quad \quad} \text{ 1. енед.}$

next:

$\text{cmp rcx, rg}$

$\text{rcx == n ?}$

минимум до края нуля

$\text{add r10, 4} \rightarrow \text{r10 = 2. енед.}$

(изменение r10 за 4 шага)

$\text{maxss xmm0, [r10]}$

↓  
разница максимального шага изменения  
агрегата отработала

$\text{add rcx, 1}$

$\text{jmp next}$

$\left\{ \begin{array}{l} \text{fxrstor [rsp]} \\ \text{mov rsp, rdx} \end{array} \right.$

done:

$\text{movss [r8], xmm0}$

$\text{leave}$

$\text{ret}$

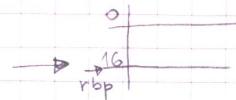
$\left\{ \begin{array}{l} \text{not supported.} \\ \text{mov rdi, 1} \\ \text{call exit} \end{array} \right.$

tui reg sse

## 2. максимальное параллелизм

maximum:

$\text{enter 16, 0}$



за 16 дайтвса изменили  
bx вмес

$\text{mov r8, rdx} \rightarrow r8 \text{ первая арт.}$

$\text{mov rg, rdi} \rightarrow rg = n$

$\text{mov r10, rfi} \rightarrow r10 \text{ вторая арт. нуля}$

$\text{mov rax, 1}$

$\text{cpuid}$

$\text{test rdx, 0x2000000}$

$\text{jz not_supported}$

$\text{mov rdx, rsp}$

$\text{and rsp, 0xff...f0}$

$\text{sub rsp, 512}$

$\text{fxsave [rsp]}$

F

нап. n=7

-1.3 12.4 3.72

-6.78 0 0.72

$\text{mov rcx, rg} \rightarrow \text{rcx = n = 7} \quad 11.9$

$\text{movss xmm0, [r10]} \rightarrow \text{xmm0 = } \boxed{\quad \quad \quad} \boxed{-1.3}$

$\text{shufps xmm0, xmm0, 0b00_000000}$

$\Gamma \text{ xmm0 } \boxed{+e \text{ умнож. } -1.3}$

00

$\rightarrow \text{xmm0 } \boxed{-1.3} \boxed{-1.3} \boxed{-1.3} \boxed{-1.3}$

$\text{add r10, 4} \rightarrow [r10] = 12.4 \text{ иж. инег. енед. нуля}$

$\text{sub rcx, 1} \rightarrow \text{rcx = 6}$

$\left\{ \begin{array}{l} \text{cmp rcx, 0} \\ \text{je done} \end{array} \right\} \text{ минимум до края нуля}$

next-four:

$\text{cmp rcx, 4} \quad \text{rcx < 4}$

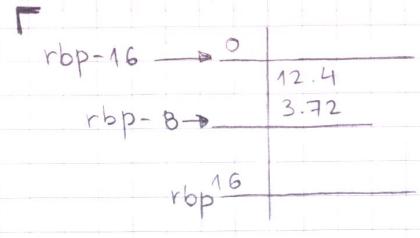
$\text{jb нема. сдвиги}$

$\text{mov rax, [r10]} \rightarrow \text{rax = } \boxed{12.4} \boxed{3.72}$

32b 32b

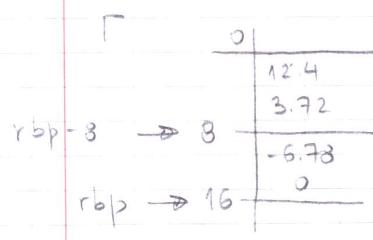
яп. сдвиги  
y 64 битами  
a 16 битами  
dp y 32b

mov [rbp-16],rax



mov rax, [r10+8] → y rax сменяется  
содержимым 64 б.  
вн. сменяется 2 едем.  
r10 + r10+4  
r10+8 -1.3 12.4 3.72  
-6.78 0 9.72  
11.9 ↑ ↑  
rax = -6.78 0 r10+12 r10+16

mov [rbp-8],rax



movups xmm1, [rbp-16] → y 128 байтами.  
xmm1 сменяется  
128 байтами = 93.11.  
содержимым  
rbp-16

$$xmm1 = \underline{12.4}, \underline{3.72}, \underline{-6.78} 0$$

maxps xmm0,xmm1

$$\begin{array}{l} \Gamma \\ \begin{array}{l} xmm0 \quad \underline{-1.3}, \underline{-1.3}, \underline{-1.3}, \underline{-1.3} \\ xmm1 \quad \underline{12.4}, \underline{3.72}, \underline{-6.78} 0 \\ \rightarrow \quad \underline{12.4}, \underline{3.72}, \underline{-1.3} 0 \end{array} \end{array}$$

add r10,16 → в результате 4 элемента  
(16 байтами)

$$[r10] = 9.72$$

sub rcx, 4 → rcx = 2  
jmp next-fair

nema-letizi:

cmp rcx, 0  
je done

maxss xmm0,[r10]

$$\begin{array}{l} \Gamma \\ \begin{array}{l} xmm0 \quad \underline{12.4}, \underline{3.72}, \underline{-1.3} 0 \\ [r10] \quad \underline{9.72} \end{array} \end{array}$$

$$\rightarrow \quad \underline{xmm0} \quad \underline{12.4}, \underline{3.72}, \underline{-1.3} \underline{9.72}$$

$$\begin{array}{l} add r10,4 \rightarrow [r10] = 11.9 \\ sub rcx, 1 \rightarrow rcx = 1 \end{array}$$

jmp nema-letizi → находит 1 краина  
xmm0 12.4 3.72 -1.3 11.9  
rcx = 0

done:

movhlps xmm1,xmm0

$$\begin{array}{l} \Gamma \\ \begin{array}{l} xmm1 \quad \underline{12.4}, \underline{3.72}, \underline{-6.78} 0 \\ xmm0 \quad \underline{12.4}, \underline{3.72}, \underline{-1.3} \underline{11.9} \end{array} \end{array}$$

$$\rightarrow \quad \underline{xmm1} \quad \underline{12.4}, \underline{3.72}, \underline{12.4}, \underline{3.72}$$

80 б.  
двоич.

иже групп  
(иже двоичные ячейки обе)

иже га  
группы  
и

maxps xmm0,xmm1

$$\begin{array}{l} \Gamma \\ \begin{array}{l} xmm0 \quad \underline{12.4}, \underline{3.72}, \underline{-1.3}, \underline{11.9} \\ xmm1 \quad \underline{12.4}, \underline{3.72}, \underline{12.4}, \underline{3.72} \\ \rightarrow \quad \underline{ymm0} \quad \underline{12.4}, \underline{3.72}, \underline{12.4}, \underline{11.9} \end{array} \end{array}$$

иже маже оставало же  
иже группе идентификатор  
иже 12.4 и 11.9

movups xmm1, xmm0  $\rightarrow$  xmm1 = 12.4, 3.72, 12.4, 11.9  
Kontrolni xmm0

shufps xmm1, xmm1, 0b01010101 ysunarac vam  
obj jep

$$\Gamma \quad \text{xmm1} = \underbrace{12.4}_{\text{započet}} \underbrace{12.4}_{\text{započet}} \underbrace{12.4}_{\text{započet}} \underbrace{12.4}_{\text{započet}}$$

maxss xmm0, xmm1

$$\Gamma \quad \text{xmm0} = \underbrace{12.4}_{\text{započet}} \underbrace{3.72}_{\text{započet}} \underbrace{12.4}_{\text{započet}} \underbrace{12.9}_{\text{započet}}$$

movss [r8], xmm0

$$\Gamma \quad [r8] = 12.4$$

započet mu je  
sam tražeći rezultat  
meni maxima učinak  
12.4 je najveći rezultat

fxrstor [rsp]

mov rsp, rdx

leave

ret

not-supported:

mov rdi, 1

call exit

## V teorema

13. void simpson(double a, double b, int n, double \*r);  
SSE2

$$f(x) = x^3, x \in [a, b]$$

$$\int_a^b f(x) dx \approx \frac{h}{3} (f(a) + 4 \sum_{i=1}^{n-1} f(a + (2i-1)h) + 2 \sum_{i=1}^{n-1} f(a + 2ih))$$

main.c :

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
```

```
extern void simpson(double a, double b, int n, double *r);
```

```
int main() {
    int n;
    double a, b, *r;
```

```
scanf("%lf %lf %d", &a, &b, &n);
```

```
assert(n > 0);
simpson(a, b, n, r);
```

```
printf("Vrednost integrala je : %lf\n", *r);
```

```
return 0;
}
```

simpson.s :

.intel\_syntax noprefix

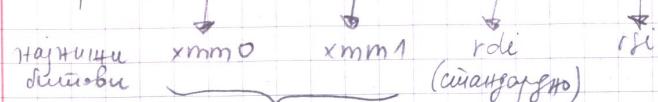
.data

.text

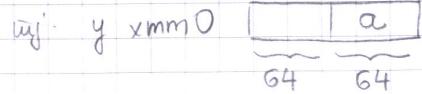
.global simpson

simpson:  
enter 0,0

simpson (double a, double b, int n, double \*r)



jep ce double  
братческие сдвиги  
указатели



установка:  
где итерации должны идти

$$4f(a + (2i-1)h) | 2f(a + 2i h) \rightarrow \text{передаётся}$$

наличие

4a + 60

$$\text{запись : } 4f(a + h) | 2f(a + 2h)$$

mov r8,rdi  $\rightarrow r8 = n$   
mov rg,rsi  $\rightarrow rg = r$

mov rax,1  
cpuid  
test rdx,0x2000000  
jz not-supported

mov rdx,rsp  
and rsp,0xffff fo  
sub rsp,512  
fxsave [rsp]

xorpd xmm8,xmm8  $\rightarrow xmm8 [0|0]$

shufpd xmm1,xmm1,0b00  $\rightarrow y xmm1 jे [hex] 6$

caga je y xmm1 [6|6]

shufpd xmm0,xmm0,0b00  $\rightarrow xmm0 [a|a]$

mov rax,1

cutsi2sd xmm2,rax  $\rightarrow$  кон. ведет к возвращению  
результату (double)  
и суммой xmm2

xmm2 [1.0]

shufpd xmm2,xmm2,0b00  $\rightarrow xmm2 [1.0|1.0]$

mov rax,2

cutsi2sd xmm3,rax  $\rightarrow xmm3 [2.0]$

mulsd xmm2,xmm3  $\rightarrow xmm2 [1.0|2.0]$

canos

нажимаю

shufpd xmm3,xmm3,0b00  $\rightarrow xmm3 [2.0|2.0]$

movupd xmm10,xmm3  $\rightarrow xmm10 [2.0|2.0]$

movupd xmm4,xmm2  $\rightarrow xmm4 [1.0|2.0]$

mulpd xmm4,xmm3  $\rightarrow xmm4 [2.0|4.0]$

запись

shufpd xmm4,xmm4,0b01  $\rightarrow xmm4 [4.0|2.0]$

mov rcx,r8  $\rightarrow rcx = n$   $\rightarrow$  xmm4 [4.0|2.0]

# h =  $\frac{b-a}{2n}$  передаётся на рабочую h как

movsd xmm5,xmm1  $\rightarrow xmm5 [1.0]$

subsd xmm5,xmm0  $\rightarrow xmm5 [b-a]$

cutsi2sd xmm6,rdi  $\rightarrow xmm6 [n.0]$

divsd xmm5,xmm6  $\rightarrow xmm5 [1.0]$

mov rax,2

cutsi2sd xmm6,rax  $\rightarrow xmm6 [1.0|2.0]$

divsd xmm5,xmm6  $\rightarrow xmm5 [1.0|2.0]$

shufpd xmm5,xmm5,0b00  $\rightarrow$  xmm5 [h|h]

next:

cmp rcx,0

je done

movupd xmm3,xmm2  $\rightarrow xmm3 [1.0|2.0]$

mulpd xmm3,xmm5  $\rightarrow xmm3 [1.0|2.0]$

addpd xmm3,xmm0  $\rightarrow xmm3 [a+10h|a+20h]$

сам передаётся  
все в один вид тайтана

movupd xmm9, xmm3

mulpd xmm3, xmm9  $\rightarrow$  xmm3  $(a+b)^2$   $(a+2b)^2$

mulpd xmm3, xmm9  $\rightarrow$  xmm3  $(a+b)^3$   $(a+2b)^3$

mulpd xmm3, xmm4  $\rightarrow$  xmm3  $4f(a+b) | 2f(a+2b)$

addsd xmm8, xmm3  $\rightarrow$  xohemo ja uvedayimoy  
peremnay za peremnay

shufpd xmm3, xmm3, 0b11  $\rightarrow$  mezhymm obje ce  
vyschi cadyalosce sano  
na najmnitni nizobedca  
xmm3 [4f | 4f]  $\rightarrow$  xmm8 [0 | 2f(a+2b)]

addsd xmm8, xmm3  
 $\downarrow$   
xmm8 [0 | 4f(a+b) + 2f(a+2b)]

sub rcx, 1

addpd xmm2, xmm10  $\rightarrow$  xmm2 [30 | 40]

jmp next  $\downarrow$   
mutoitu ce ca R

done:  $\rightarrow$  usmano oite 2 cyne, jauj je ocuvano go gosano

movupd xmm9, xmm0  $\rightarrow$  xmm9 [a | a]

mulpd xmm0, xmm9  $\rightarrow$  xmm0 [a<sup>2</sup> | a<sup>2</sup>]

mulpd xmm0, xmm9  $\rightarrow$  xmm0 [a<sup>3</sup> | a<sup>3</sup>]

addsd xmm8, xmm0  
 $\rightarrow$  xmm8 [0 | f(a) + 4f(a+2) + 2f(a+4)]

movupd xmm9, xmm1

mulpd xmm1, xmm9

mulpd xmm1, xmm9  $\rightarrow$  xmm9 [b<sup>2</sup> | b<sup>2</sup>]

subsd xmm8, xmm1

$\rightarrow$  xmm8 [0 | f(a) - f(b) + ]

$\frac{2}{3}$  jauj

mulsd xmm8, xmm5  $\rightarrow$  zmenitituru ca h

mov rax, 3

cvttsi2sd xmm5, rax

divsd xmm8, xmm5  $\rightarrow$  zmenitituru ca 3.0

movsd [rg], xmm8

5. void center-of-mass (int n, float\* p, float\* m, float\* c)

SSE

droj chetnaya. naz chetnaya sum masca yestvaya  
y.y.2) chetnaya mace  
(naz koordinata x.y.u.z)

ytocse negom x,y,z,m jegte chetnaya

paruyay ce  
to doplyut

$\sum_{i=0}^{n-1} m_i$

$r = \frac{1}{\sum_{i=0}^{n-1} m_i} \sum_{i=0}^{n-1} m_i \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

main.c:

extern void center-of-mass (int n, float\* p, float\* m, float\* c);

int main () {

int n; i;  
float \*p, \*m;  
float c[3];

scanf ("%d", &n);  
assert (n > 0);

assert ((p = malloc (3 \* n \* sizeof (float))) != NULL);

assert ((m = malloc (n \* sizeof (float))) != NULL);

for (i = 0; i < n; i++) {  
scanf ("%f %f %f", &p[3 \* i], &p[3 \* i + 1], &p[3 \* i + 2]);  
scanf ("%f", &m[i]);

}

center-of-mass (n, p, m, c);

printf ("Center of mass im sledete koordinate:\n");  
printf ("X = %.f\n Y = %.f\n Z = %.f\n", c[0], c[1], c[2]);

free (p);  
free (m);  
return 0;

center-of-mass.s:

f(int n, float \*p, float \*m, float \*c)  
 ↓      ↓      ↓      ↓  
 rdi    rsi    rdx    rcx

$$x = \frac{\sum x_i m_i}{\sum m_i}, y = \frac{\sum y_i m_i}{\sum m_i}, z = \frac{\sum z_i m_i}{\sum m_i}$$

$$\frac{x_1 m_1}{m_1} + \frac{x_2 m_2}{m_2} + \dots$$

center-of-mass:

enter 16.0

mov rg, rdx → rg = aplica uoreuka tufa mase

mov r8, rcx → r8 = aplica pesymetria

mov r10, rsi → r10 = aplica uoreuka tufa konguracije

{  
 mov rax, 1  
 cpuid  
 test rdx, 0x2000000  
 jz not\_supported

mov rdx, rsp  
 and rsp, 0xffff...f0  
 sub rsp, 512  
 fxsave [rsp]

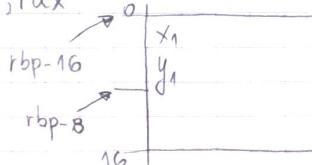
xorps xmm2, xmm2 → xmm2 [0 0 0 0]

xorps xmm3, xmm3

mov rcx, rdi → rcx=n

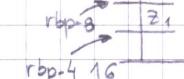
next one:

mov rax, [r10] → rax [x1 | y1]  
 mov [rbp-16], rax



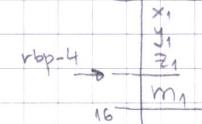
mov eax, [r10+8] → [x1, y1, z1, x2, ...]  
 ↓  
 8 bajta + 8a = 64 bajta  
 r10+8  
 eax [z1]

mov [rbp-8], eax → [x1, y1, z1]



mov eax, [rg] → eax [m1]

mov [rbp-4], eax → [x1, y1, z1, m1]



movups xmm0, [rbp-16] → xmm0 [m1 | z1 | y1 | x1]

addps xmm2, xmm0 → xmm2 = xmm2 + xmm0

movups xmm1, xmm0

shufps xmm1, xmm1, 0b11111111 → xmm1 [m1 | m1 | m1 | m1]

mulps xmm0, xmm1 → xmm0 [m1^2 | z1m1 | y1m1 | x1m1]

addps xmm3, xmm0 → xmm3 = [m1^2 | z1m1 | y1m1 | x1m1]

add r10, 12 → tuncirano r10 za 3 enene. [r10] = x2

add rg, 4 → tuncirano rg za 1 enene. [rg] = m2

dec rcx

cmp rcx, 0

jne next-one.

# Kada rcx je vec 0, ciscim da ga upozaj, vise od faziranja  
 cisc da se smanjene u y permutacija je ceze te ciscu

xmm3 [Σmi^2 | Σzimi | Σyimi | Σximi]  
 xmm2 [Σmi | Σzi | Σyi | Σxi]

movups xmm0, xmm3

shufps xmm2, xmm2, 0b11111111 → xmm2 [Σmi | Σmi | Σmi | Σmi]

rcpps xmm2, xmm2 → xmm2 [1/Σmi | 1/Σmi | 1/Σmi | 1/Σmi]

mulps xmm0, ymm2  $\rightarrow$  xmm0

$\sum_{mi}^2$	$\sum_{mi}^{2i}$	$\sum_{mi}^y$	$\sum_{mi}^{mix}$
$\sum_{mi}$	$\sum_{mi}$	$\sum_{mi}$	$\sum_{mi}$

movss [r8], xmm0  $\rightarrow$  [r8] =  $\frac{\sum_{mix}}{\sum_{mi}}$

shufps xmm0, xmm0, 0b11100001  $\rightarrow$  odpisano do cache'a

movss [r8+4], xmm0  $\rightarrow$  [r8+4] =  $\frac{\sum_{mi}^y}{\sum_{mi}}$

shufps xmm0, xmm0, 0b11100010

$\sum_{mi}^2$	$\sum_{mi}^{2i}$	$\sum_{mi}^y$	$\sum_{mi}^{mix}$
$\sum_{mi}$	$\sum_{mi}$	$\sum_{mi}$	$\sum_{mi}$

movss [r8+8], xmm0  $\rightarrow$  [r8+8] =  $\frac{\sum_{mi}^{2i}}{\sum_{mi}}$

{ fxrstor [rsp]  
mov rsp, rdx

done: leave  
ret

not-supported:

movl d1, 1  
call exit

## VI Negesva

15.

void quadratic\_form(int n, float\*\* a, float\*x, float\*f)  
SSE

$$f = (x_1 \ x_2 \ \dots \ x_n) \cdot \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} =$$

$$= x_1(a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n) + x_2(a_{21}x_1 + \dots + a_{2n}x_n) + \dots + x_n(a_{n1}x_1 + \dots + a_{nn}x_n)$$

main.c:

extern void quadratic\_form(int n, float\*\* a, float\*x, float\*f);

int main(){

int i, j, n;

float \*\*a, \*\*x, f;

//uvravabano ynas us qajna

FILE \*fajl = fopen ("input.txt", "r");

fscanf (fajl, "%d", &n);

assert (n>0);

assert ((a = malloc (n \* sizeof (float\*))) != NULL);

for (i=0; i<n; i++) {

assert ((a[i] = malloc (n \* sizeof (float))) != NULL);

for (j=0; j<n; j++)

fscanf (fajl, "%f" &a[i][j]);

}

assert ((x = malloc (n \* sizeof (float))) != NULL);

for (i=0; i<n; i++)

fscanf (fajl, "%f", &x[i]);

quadratic\_form (n, a, x, &f);

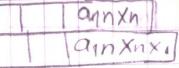
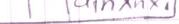
free (x);

for (i=0; i<n; i++)

free (a[i]);

free (a);



$\text{mulss xmm0, xmm1} \rightarrow \text{xmm0}$    
 $\text{mulss xmm0, xmm2} \rightarrow \text{xmm0}$    
 $\text{addss xmm3, xmm0} \rightarrow \text{xmm3}$   

$$[a_{14}x_4x_1 + a_{18}x_8x_1 + \dots + a_{13}x_3x_1 + \dots]$$
  

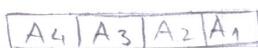
$$\dots | a_{12}x_2x_1 + \dots | a_{11}x_1^2 + \dots + a_{nn}x_n^2]$$

$\text{add rsi, 4}$   
 $\text{add r8, 4}$   
 $\text{dec rcx}$   
 $\text{jmp next-one}$

continue:  $\rightarrow$  битану ја сако одјасните 1 рег  
 сак употреба митотиците све  $(A X_2)$  иштв.  
 $\text{add r10, 8}$   $\rightarrow$  почетарно доделите вредноста  
 за 1 инд јер је веќи вис  
 у индексијата која је у Intel 64  
 $64\text{десетка} = 8$  десетка

$\text{pop r8}$  } јеп митотиците све индексите  
 $\text{pop rcx}$  } са  $X_2$ , па вако сака  $X_3$  иштв

loop next-row  $\rightarrow$  употребијте  $rcx$  за 1, ако је  $rcx = 0$   
 употреби  $rcx$  за 1, ако је  $rcx > 0$   
 употреби  $rcx$  за 1, ако је  $rcx < 0$

$\Gamma$   $\text{xmm3 }$  

сак употреба ја садржатење објекта 4

$\text{movhlps xmm2, xmm3} \rightarrow$   
 $\text{xmm2 } \begin{matrix} \text{[} a_{11} \text{]} & \text{[} a_{12} \text{]} \\ \text{[} a_{13} \text{]} & \text{[} a_{14} \text{]} \end{matrix} \quad \text{A}_3 \quad \text{A}_4$

$\text{addps xmm3, xmm2} \rightarrow$   
 $\text{xmm3 } \begin{matrix} \text{[} a_{11} \text{]} & \text{[} a_{12} \text{]} \\ \text{[} a_{13} \text{]} & \text{[} a_{14} \text{]} \end{matrix} \quad \text{A}_2 + \text{A}_4 \quad \text{A}_1 + \text{A}_3$

$\text{movaps xmm2, xmm3}$   $\overset{01}{\rightarrow}$   
 $\text{shufps xmm2, xmm2, 0b01010101}$   
 $\rightarrow \text{xmm2 } \begin{matrix} \text{[} a_{11} \text{]} & \text{[} a_{12} \text{]} \\ \text{[} a_{13} \text{]} & \text{[} a_{14} \text{]} \end{matrix} \quad \text{A}_2 + \text{A}_4$

$\text{addss xmm3, xmm2} \rightarrow \text{xmm3 } \begin{matrix} \text{[} a_{11} \text{]} & \text{[} a_{12} \text{]} \\ \text{[} a_{13} \text{]} & \text{[} a_{14} \text{]} \end{matrix} \quad \text{A}_1 + \text{A}_2 + \text{A}_3 + \text{A}_4$

$\text{movss [r15], xmm3}$   
 $\downarrow$   
 $\text{migi}$   
 $\text{migra}$   
 $\text{migr}$

$\left\{ \text{fxrstor [r15]}$   
 $\text{movl r15, dx}$

done:  
 leave  
 ret  
 not-supported:  
 $\text{movl rdi, 1}$   
 $\text{call exit}$

\*gdb:  
 1. пребогнуво програма со -g гогашум точе gcc  
 2. gdb a.out  
 3. tui enable  
 4. tui reg general  
 5. start  
 6. focus regs  
 7. next / step  $\rightarrow$  упаку у објект кој е интересантен  
 8. q  $\rightarrow$  упаку

VII Недеља 20.11.2019.

ARM (32 бити)

- подаци величине 4 байта морају бити на адреси деливој са 4
- подаци величине 2 байта морају бити на адреси деливој са 2
- инструкције су увек на адреси деливој са 4

16 32-битних регистара r0 - r15

- program counter (pc) регистар
- први и последњи регистар у овом регистару вредност се смене са следећим адресама
- први и последњи регистар вредностима овог регистара добијају се адресе које наведене инструкције користе

→ \* инструкције

• Load/store инструкције

- остале инструкције не могу имати паметијске операнде (све ових врса)

• учитавање из паметије:

- ldr → учитава 1 32-битни податак
- ldrb → учитава 1 байт, а остатак регистара је њуне
- ldrh → учитава 16-битни податак, остатак је 0
- ldrsb → учитава 1 байт, остатак је битови знаци
- ldrsh → учитава 10-битни податак, остатак је бити знаци

• кување у паметију:

- str → копира 32-битни податак у паметију
- strb → копира 1 байт (најмање паметије)
- strh → копира 16-битова (чињеница)



ARM Load/store агреси ногови:

- offset: битни регистар + offset који може дати +/- 12 битната константа, или +/- други регистар, који обично може дати шифрован за константама број података

нпр. ldr r0, [r1, #4] → учитава реч са адресе r1+4  
ldr r0, [r1, #-8] → учитава са адресе r1-8  
str r0, [r1, r2] → кува вредност r0 на адресу r1+r2  
str r0, [r1, -r2, lsl #2] → кува вредност r0 на адресу r1-4\*r2

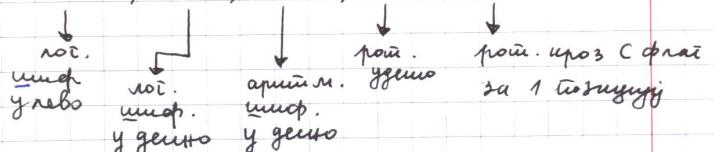
- pre-indexed: чини насе и offset само чини се битни регистар ажурира на изразнуту адресу

нпр. ldr r0, [r1, #8]! → учитава реч са адресе r1+8 а затим регистар r1 поставља на r1+8

- post-indexed: чини насе и offset само чини се битни реч ажурира на начин чини се њенова ординација вр. употреби насе адреса

нпр. str r0, [r1], #4 → кува r0 на адресу r1 а затим уставља r1 за 4

- шифреци: lsl, lsr, asr, ror, rrz



• инструкције за одрагу података

add	+ одредиште + source + source
adc (сабираше са претосом)	регистар операнд операнд
sub	
8bc (одузимаше са поузадијем)	
r8b (однужно одузимаше)	
rsc	
and (конјункција)	
(xor)	
eor (or)	

уvec  
регистар shifter  
операнд

cmp (упоредивање одузимањем)	+ регистар + shifter
cmin (упоредивање сабирањем)	операнд
tst (инцирирање битовима и појм.)	
teq (инцирирање xor-ом)	

регистар + shifter  
операнд

mov	+ регистар + shifter
mvn (пременитеље уз престо. комплементарне битове)	операнд

регистар + shifter  
операнд

shifter операнд је:

- 32 битна компонента
- регистар
- импримован рес.

нпр.

add r0, r0, #1 → r0 = r0 + 1  
 sub r0, r1, r2 → r0 = r1 - r2  
 r8b r0, r1, r2, lsl #1 → r0 = 2 \* r2 - r1  
 and r0, r1, r2, lsl r3 → r0 = r1 & (r2 \* 2^r3)

mvn r0, r0 → r0 = ~r0  
 r8b r0, r0, #0 → r0 = -r0  
 mov r0, r0, lsl #2 → r0 = r0 \* 4

• инструкције линије

mul → има 3 операнда и сви су регистри;  
 линија друга 2 и следећа низних 32 б  
 у први операнд.

imull → има 4 операнда. 1. оз. је одредиште  
 низних 32 б, а 2. висих 32 б,  
 3. и 4. су чиниоци ; неознакено

mla → 4 операнда, линији 2. и 3. и сабира са 4.  
 и низних 32 б следећа 1.

Smull → ограничено линије

umlal

smlal

• инструкције транзија

једна се преводи у рен. адресу  
 у формату адресу pc

bl → иницијал и в са којој адресију инструкције

која следи наимен bl инструкције следица је r14  
 као call у Intel64

bl  
регистар

• условно извршавајуће инструкције

- на име инструкције се добију eq, ne, gt, lt, ge, le, hi,  
 lo, hs, ls, ...

нпр. addgt r0, r0, r1 → r0 = r0 + r1 само ако је  
 претходно дима беће

subeq r1, r2, r3 → r1 = r2 - r3 само ако је  
 претходно дима једнако

mult r1, r2, r3 → r1 = r2 \* r3 само ако је  
 претходно дима највеће

### • условни команди

bgt lne bhi (если аргумент больше, чем значение)  
 beg lbt blgt (если аргумент меньше, чем значение, то возврат  
аргумента в lr при)

### • опционо ампирираше директива

- на инструкцији се дода S, ако аргумент је по умолчанију  
не ампирираше директива

инструкције за учитавање и чување стека регистара

ldm → учитава из паморије бр. рег. из стека  
чији

stm → чува у паморију бр. рег. из стека чији

- имају суфикс који одређује смерни ног:

- IA (Increment after) → значи да се користи след. ног.

$Rb, Rb+4, Rb+8, \dots, Rb+4^{*(k-1)}$

↓  
 данас  
 ре.  
 ↓  
 овој ре.  
 ујасни  
 чији

- IB (Increment before) →  $Rb+4, Rb+8, \dots, Rb+4^* k$

- DA (Decrement after) →  $Rb, Rb-4, Rb-8, \dots, Rb-4^{*(k-1)}$

- DB (Decrement before) →  $Rb-4, Rb-8, \dots, Rb-4^* k$

нпр.

ldmia r0, {r1, r2, r3} → M = [r0], r2 = [r0+4], r3 = [r0+8]

ldmdb r0, {r0, r4, r2} → r4 = [r0-4], r2 = [r0-8], r0 = [r0-12]

stmib r1, {r0, r1} → [r1+4] = r0, [r1+8] = r1

stmda r0, {r1, r2, r3} → [r0] = r3, [r0-4] = r2, [r0-8] = r1

- ако се иза датог регистара стави узвитник, тада  
се дати регистар напон одављае обрачујући  
ампирираше чији се за 4\*к датијева (у случају  
је ивица ib), али се умножује за 4\*к (da, db).

нпр.

ldmia r0!, {r1, r2} → r1 = [r0], r2 = [r0+4]  
r0 = r0+8

stmda r0!, {r1, r2} → [r0-4] = r2, [r0-8] = r1  
r0 = r0-12

- алигернативни суфикс (примеђени сите су обрачунати):

fd (Full descending stack)

ed (Empty desc. stack)

fa (Full ascending stack)

ea (Empty asc. stack)

потпуности у односу на ia, ib, da, db :

ldmfa = ldmda

ldmfd = ldmia

ldmea = ldmdb

ldmed = ldmib

stmed = stmda

stmea = stmia

stmfd = stmdb

stmfia = stmib

нпр. { stmfld sp!, {r0, r1} } → мапирају на стек  
r0 ur1

{ ldmfld sp!, {r0, r1} } → стек са стеком  
r0 ur1

\* сачијавање унутре регистара

r15 → pc

r14 → lr

r13 → sp (stack pointer)

r12 → ip (intra-procedure-call scratch register)

r11 → fp (stack frame pointer)

→ r0, r1, r2, r3 претпостављају коришћеног рејса (у тој повратници  
бр. 1).



## II Начало

снр r0,r1  
movlt r0,r1

sudo mount -o loop  
armedslack.img tmp/  
sudo umount tmp/  
  
mano ige cam  
parauobara  
BM

## VIII Регион

① fact.s:

.text  
.align 2  
.global fact

@ int fact(int n)

$\tilde{r0}$

fact:

stmfd sp!, {fp, lr}  
mov fp, sp

stmfd sp!, {r4}  $\rightarrow$  cybano r4 na cireuy jep kerosca  
использование  
mov r4, r0  $\rightarrow$  r4 = r0 = n

@ usnas us neupryye je fact col=1

cmp r0, #0  
bne rec.call  $\rightarrow$  aus r0 ≠ 0 из-за неуправления  
@ a where r0 je r0 #0 в батарею 1  
mov r0, #1  
b done

rec-call:

@iuzubano fact(n-1)

sub r0, r0, #1

bl fact

@ nirovunuo n \* fact(n-1)

mov r2, r0  $\rightarrow$  r0 je из-за языка Bp fact  
mul r0, r2, r4  $\rightarrow$  r0 = r2 \* r4

done:

ldmfd sp!, {r4}

mov sp, fp

ldmfd sp!, {fp, pc}

## II. Homework

@ упаковка  $1 \times 2 \times \dots \times n$  в массивы  $r3$ ;  $1 \leq i \leq n-1$   
 $\text{mov } r3, \#1$

loop:

$\text{cmp } r0, \#0$   
 $\text{beq end\_loop}$

$\text{mov } r2, r3$   
 $\text{mul } r3, r0, r2$

$\text{sub } r0, r0, \#1$

loop

end\_loop:

$\text{mov } r0, r3$

②

linsearch.s:

@ int linsearch (int \*a, int n, int x)  
 $\underbrace{\quad}_{r0} \underbrace{\quad}_{r1} \underbrace{\quad}_{r2}$

linsearch:

$\text{stmfd } sp!, \{fp, pc\}$   
 $\text{mov } fp, sp$

$\text{str } r1, [sp, \#-4]$  → убираю  $r1$  на стек  $sp$ .  $n$

loop:

$\text{cmp } r1, \#0$   
 $\text{beq not\_found}$

у когд  $a[r1]$   $\neq x$  идем к следующему элементу

$\text{ldr } r3, [r0], \#4$  → упаковываю следующий элемент  $y$  в  $r3$

$\text{cmp } r3, r2$   
 $\text{beq found}$

} упаковка  $y$

@ упаковка

$\text{sub } r1, r1, \#1$   
 loop

found:

устанавливаем  $r1$  на конец стека  
 @ возвращение из функции в функцию  
 $\text{ldr } r2, [sp], \#4$  →  $jrp$  возвращает значение  $sp$   
 $fa - 4$

$\text{slub } r0, r2, r1$

loop

not-found:

$\text{mov } r0, \#-1$

loop:

↓  $\text{ldmfd } sp!, \{fp, pc\}$   
 $\text{mov } sp, fp$

③

max\_element.s:

@ int max\_element (int  $\underbrace{a[\cdot]}_{r0}, \underbrace{n}_{r1})$

max\_element:

... (упоряд.)

$\text{ldr } r3, [r0], \#4$

$\text{sub } r1, r1, \#1$

loop:

$\text{cmp } r1, \#0$   
 $\text{beq end\_loop}$

$\text{ldr } r2, [r0], \#4$

$\text{cmp } r2, r3$  }  $a[r2] > r3$   $r3$   $\leftarrow r2$   
 $\text{movgt } r3, r2$  }  $y \leftarrow r3$

$\text{sub } r1, r1, \#1$

b loop

end-loop:  
mov r0, r3

(4)

minimax.s:

@ void minimax ( $\underbrace{\text{int } a[]}_{r0}$ ,  $\underbrace{\text{int } n}_{r1}$ ,  $\underbrace{\text{int } \star \text{min}}_{r2}$ ,  $\underbrace{\text{int } \star \text{max}}_{r3}$ )

minimax:

...  
stmfd sp!, {r4-r6}  $\rightarrow$  y r5 korus ybasa  
max ay r4 min

[ldr r5, [r0], #4]  $\left\{ \begin{array}{l} \text{ta. gauj u min - max} \\ \text{1. eadu.} \end{array} \right.$   
mov r4, r5  
sub r1, r1, #1

loop:

cmp r1, #0  
beq end-loop

@ y r6 yuvarno meyhtu enere.

[ldr r6, [r0], #4]

cmp r6, r5  $\rightarrow$  yuopetgyimota ca max

movgt r5, r6  $\rightarrow$  awo jf betu of meeraong s  
jf awo uotb near

cmp r6, r4

movlt r4, r6  $\rightarrow$  awo jf meeku of min  
0+ga jf awo nobu min

sub r1, r1, #1

b loop

end-loop:

str r4, [r2]  
str r5, [r3]

ldmfd sp!, {r4-r6}

(5)

arm-strlen.s:

@ int arm-strlen( $\underbrace{\text{char } \star s}_{r0}$ )  $\rightarrow$  yuifagymuuy cymuas s

arm-strlen:

...  
mov r1, r0  $\rightarrow$  meyhtu napaucep y5  
mov r0, #0  $\rightarrow$  sponal napaucep y5

next-char:

[ldrb r2, [r1], #1]  $\rightarrow$  yuialans 1 dajā  
uj. 1 napaucep os s y12

cmp r2, #0  
beq done

add r0, r0, #1  
b next-char

(6)

sum.s:

@ int sum( $\underbrace{\text{int } a[]}_{r0}$ ,  $\underbrace{\text{int } n}_{r1}$ );

sum:

...  
str r4, [sp, #-4]!

mov r2, #0  $\rightarrow$  i=0  
mov r3, #0  $\rightarrow$  S=0

## IX Regresa

loop:  
 cmp r2, r1      } i < n?  
 bge end-loop  
 ldr r4, [r0, r2, a[r #2]]      r4 = r0 + 4 \* r2  
 add r3, r3, r4      S = S + r4      inicializar  
 add r2, r2, #1      suma  
 b loop

end-loop:  
 mov r0, r3

done:  
 ldr r4, [rsp], #4

II Haran

mov r3, #0

loop:  
 cmp r1, #0  
 bge end-loop

ldr r2, [r0], #4  
 add r3, r3, r2  
 sub r1, r1, #1  
 b loop

end-loop:  
 mov r0, r3

III Haran

mov r3, #0

loop:  
 cmp r1, #0  
 ldrihe r2, [r0], #4  
 addhe r3, r3, r2  
 subne r1, r1, #1  
 one loop

① binsearch .S:      para la búsqueda en un array de 4x3  
 @ int binsearch (int\*a, int n, int x)  
 r0      a  
 r1      n  
 r2      x  
 binsearch:  
 ...  
 strndl sp!, {r4, r5}

mov r4, #0      señal variable l = 0  
 sub r5, r1, #1      getchar unsigned d = n - 1  
 loop:  
 @ checa si l <= d  
 cmp r4, r5  
 bgt not-found

@ Mary Haran chequea si existe S =  $\frac{l+d}{2}$   
 add r1, r4, r5

mov r1, r1, asr #1  
 kao ga una condición es 2 <= d & d <= n  
 una condición que cumple do l <= d & d <= n.

@ Mary Haran busca el elemento a[S]  
 ldr r3, [r0, r1, asl #2]

r0 + 4 \* r1  
 a      S

cmov r2, r3  
 beq found

@ Mary Haran si no existe  
 addgt r4, r1, #1      actualizar l = d + 1  
 sublt r5, r1, #1      actualizar r2 = r0 + l  
 adgrievy, aij, reba  
 waarej, aij, reba  
 l = S + 1

ano if r2 < r3, r1 ve  
 Haran y return nothing  
 u getchar unsigned d = S - 1  
 b loop

found:

mov r0, r1  
b done

not-found:

mov r0, #1

done:

ldmfd sp!, {r4, r5}

② euler.s:

@ int euler(int n)

r0

ogrejyj spoj yevena op. m  
ing. cy yzajansu urocen  
ea n m 15man

euler:

slmfld sp!, {r4-r6}

mov r6, #0  $\rightarrow$  spojec yzajansu urocen  $\frac{1}{n}$

mov r5, r0  $\rightarrow$  r5 = n

mov r4, #1  $\rightarrow$  inuytu spoj m so uoya urocen  
ganujc yzajansu urocen

next-number:

cmp r4, r5  $\rightarrow$  auo ji m = n

beq last-number

r6 r5  
" " "  
r0 r1  
" " "

mov r0, r5 } urobansu opjj nad (m m)  
mov r1, r5 }  
bl nzd m

@auo ji nzd(n,m)=1  $\rightarrow$  r6++  
cmp r0, #1  $\rightarrow$  r0 ji urobansu opjj ofjejed  
add eq r6, r6, #1

@uvare ujosef ebanosa eneg. opjj  
add r4, r4, #1  
b next-number

last-number:

mov r0, r6

ldmfd sp!, {r4-r6}

... (cūuoi obj)

@ obja int nzd(int n, int m)

r0 r1

nzd:

slmfld sp!, {r4-r5}

mov r4, r0  $\rightarrow$  r4 = n

mov r5, r1  $\rightarrow$  r5 = m

@ Eyunugob ani.

next-iteration:

@ auo ji m=0 otagaf n nzd

cmp r5, #0

beq last-iter

mov r0, r4 } urobansu opjj mod(n,m)  
mov r1, r5 }  
(\*) bl mod

mov r4, r5  $\rightarrow$  n = m

mov r5, r0  $\rightarrow$  m = n \* m

b next-iteration

last-iter:

mov r0, r4

ldmfd sp!, {r4, r5}

@ int mod r1=rn, rn=r1

mod:

@ yrciuonoffgnanruo mof n jocneojf  
n, m

next-subtraction:

cmp r0, r1

subge r0, r0, r1

bge next-subtraction

II Hanuu je des obje mod

(\*) ymcuuo obe hageye je bl — modsi 3

oba obja  
offgtyje n, m

③ palindrom.s:

@ int palindrom (char \*s)  
r0

palindrom:

{ stmfld sp!, {sp,lr}  
mov fp, sp

mov r1, r0 → yrl je cap aqpeca boleimn eipunra  
ja offgja ja boleimn ja boleimy  
ja boleimbu kapamdep

next-char: → obaciuuvalanu aonepa r1 ga iacuuy  
ymlasba ldrb r2, [r1] → r2=1. cap ja kpey  
ymlasba

cmp r2, #0 → obepcal a mafajmif  
bea last-char 1. kaf = \0

add r1, r1, #1  
b next-char

last-char:

sub r1, r1, #1 → ga +1 he dyge \0 zero day  
tacnegru kap. cemreuta

next-pair:

cmp r0, r1 → ymcuuo 1. u tacnegru ukazubu  
bge last-pair → aus r0>r1 nema bnuo

mcuuo jaec ymcuuo

ldrbl r2, [r0], #1 → yr2 ymcuuo 1. ucp.

ldrbl r3, [r1], #1 u tmcuuo r2 za 1  
(u. ha engetu)

cmp r2, r3

bne fail → aus Hucey fgrann

o next-pair

last-pair:

mov r0, #1 → jutte manypam  
b done

fail:

mov r0, #0 → mye manypam

done:

{ mov sp, fp  
{ ldufd sp!, {fp, pc}}

④ secret.s:

@ int secret (int x)  
r0

secret:

{ ... }

stmfld sp!, {r4-r6}

mov r4, r0 → r4=x

next-number:

@ gonneiog jr dpoj gboayudppet  
cmp r4, #10  
blt last-number

mov r5, #0 → cynca yndape

next-iter:

cmp r4, #0 → gonneiog he usgbojina che yndape  
beq last-iter

uparhenu	mov r0, r4
ouzunak	mov r1, #10
uprager.	bl — mod8i3
ca10 →	
uaoje cnp.	
yndape.	add r5, r5, r0

→ gojaje no usgbojey yndape  
na cynap

mov r0, r4
mov r1, #10
bl — div8i3
mov r4, r0

b next-iter

last-iter:

mov r4, r5 → zdup yndape jr nobu dpoj  
b next-number

last-number: ieq jr dpoj ffusynd.  
mov r0, r4

[ldmfd sp!, {r4-r6}]

mov sp!, fp  
[ldmfd sp!, {fp, pc}]

x negara

① longest-s:

② void longest (char \* str, int \* start, int \* length)

apotanamu Hajgyuy ceuberry uuuux naranidep y  
cineury

C kôg du du:

i=0;  
prevChar = niz[i];  
c = niz[i];

while (c != '\0') {

if (c == prevChar)  
br++;

else {

if (br > maxBr) {

maxBr = br;

maxChar = prevChar;

}

br = 1;  
prevChar = c;

c = niz[i+1];

}

if (br > maxBr) {

maxBr = br;

maxChar = prevChar;

}

—

longest:

stmfld sp!, {fp, lr}  
mov fp, sp

stmfld sp!, {r4-r8}

mov r4, #0 → rybaleno aozemtu usfeme go capa  
Hajgyuy ceuberry

mov r5, #0 → rybaleno yndape zo capa raffym ceuberry

[ldr r3, [r0], #1 → ymirabano 1.map.

cmp r3, #0

beg store

2

mov r6, #0 → инициализация memory сешибаже

mov r7, #1 → инициализация языка сешибаже

mov r8, r3 → инициализация map.

next-char:

[ldr r3, [r0], #1

cmp r3, #0

beg last-char

cmp r8, r3 } and cy инициализация языка

beg equal

cmp r7, r5 } and je инициализация less than or equal  
ble skip case

@ and Heijo ≤

mov r4, r6

mov r5, r7

skip:

add r6, r6, r7 → r6=r6+r7

mov r7, #1

b continue

equal:

add r7, r7, #1

Continue:

mov r8, r3

b next-char

last-char:

@ ga mi je now ga исчезнова сешибажа Heijo

cmp r7, r5

ble store

mov r4, r6

mov r5, r7

store:

str r4, [r1]

str r5, [r2]

ldmfd sp!, {r4-r8}

② MIRROR.S:

@ unsigned mirror (unsigned x) →  $\underbrace{sp[0] \dots sp[31]}_{r0}$   $\underbrace{\text{для языка}}$   $\underbrace{\text{иначе}}$   $\underbrace{\text{регистры}}$   
 $\underbrace{\text{значения}}$   $\underbrace{\text{для языка}}$

MIRROR:

{ ... }

mov r3, #32 → Spojaz инициализация (for H x 32 bytes)

mov r1, #0 → обработка символов

next-bit0:

cmp r3, #0

beq last-bit0

mov r1, r1, lsl #1 → инициализация языка

1 символ

r1 =  $\underbrace{0000 \dots 0000}_{32}$

← 1

r1 =  $\underbrace{0000 \dots 0000}_{32}$

@ инициализация языка языка инициализации языка

tst r0, #1

@ and je 1 отработано языка языка языка языка

орне r1, r1, #1 → r1 = r1 || 0000 0001

ано  
Heijo

@ инициализация ячейки 141003.  
mov r0, r0, lsr #1

sub r3, r3, #1  
b next-bit

last-bit:  
mov r0, r1

{ ... }

global printbits

@ void printbits(unsigned x)

printbits:  
{ ... }

stmfd sp!, {r4, r5}

mov r4, r0

mov r5, #1  
mov r5, r5, lsl #31

macua

r5 = 1000 ... 0000

next-bit:

cmp r5, #0  
beq last-bit

r4&r5 < tst r4, r5  
and r4  
tst r4, r5  
beq zero  
per. r4, #0

r5 jf y 1. инициал. 1000 . 0000  
знач 32. инициал. инициал. 0 y r4  
онга инициал. за инициал. инициал.

инача

@ инициал. инициал. инициал. инициал.

mov r0, #1'1'

bl putchar

o continue

zero:

mov r0, #0'  
bl putchar

continue:

mov r5, r5, lsr #1 → r5 = 0100 ... 0000  
b next-bit

last-bit:

mov r0, #'n'  
bl putchar

ldmfd sp!, {r4, r5}

③ most-frequent.s:

ro      r1      r2

@ void most-frequent(char \*s, char \*c, int \*f)

ограничение короткое? но не забыть инициализации  
и инициализации f

most frequent:

{ ... }

@ инициализация массива за пределами памяти

sp. инициал. 256

256 · 4 = 1024  
↓  
дубликат int  
память

sub sp, sp, #1024

stmfd sp!, {r4-r6}

sub r3, fp, #1024

mov r4, #0 → инициал.

mov r5, #0

next: → y инициал. инициал. инициал. инициал.  
cmp r4, #256 → while r4 < 256  
beg last  
Ho: 0

str r5, [r3+r4, lsl #2] → ~~запись~~  
y элемента ~~последовательности~~  $r_3$   
~~(сдвигом  $r_4$ )~~

last:

next-chap

next\_char: (drb r5, [r0], #1 → yuntabanu 1. wyp.

cmp r5, #0 → r5 >= '\0'  
beq last-char

@awo 15 Hujjat yebhabano djojar

@ y 14 ymucabano Spojare 1cap. ca ASCII  
usogom 15 , ybetabano varsa 1 u bratans  
nemoprij

[ldr r4, [r3,r5, lsl #2]  
add r4, r4, #1  
str r4, [r3, r5, lsl #2]

b next-char

last-chak:

-char:  
@ преда нај највећи међу пројектима  
шта је, као и неке најбоље пројек-  
(које су кер. који се посматрају као  
јесу) шта

@ 14 te chura wretchen laem. Heraa ~~jetzt~~. try  
@ 16 utguge male. laem. ~~be seen~~  
→ 119 is this met

ldr r4,[r3]  
mov r6,#0

если у тебя есть  
максимально  
высокий коэффициент  
коэффициент

@ ro the Nutz oproj  
Nov 10, #1

next-element :

cmp r0,#256

beg last - element

[ldr r5, [r3,r0], lsl #2] <sup>→ ungetc</sup>

cmp r5,r4

movgt r4,r5

movgt r6, r0

add r0,r0,#1  
b next\_element

last\_element:

strb r6,[r1]

str r4, [r2]

l d m f d s p ! , { r 4 - r 6 }

## X1 Неделья

① norms.s:

@ void norms (int \*\*a, int m, int n, int \*h,  
 int \*v) арг. итд  
 int \*\*a Брояв Брояв Брояв Брояв make  
 int \*v Брояв Брояв Брояв Брояв Брояв

[fp+8]

make-  
to монотоне

norms:

{ ...

@ аналогично нок. арг. итд Брояв Брояв Брояв Брояв  
 монотонна монотонна монотонна монотонна

sub sp, sp, r2, lsl #2 максимум  
запись запись  
 stmfd sp!, {r4-r10}

@ r10 за счет agree эта строка тут  
 монотонна

sub r10, fp, r2, lsl #2

@ инициализируююю элементы нов. мас

mov r4, #0 → оправа  
 mov r5, #0

init-next: white  
 cmp r4, r2 → V i < n  
 bne init-last

str r5, [r10, r4, lsl #2] → инициализ.

add r4, r4, #1  
 b init-next

init-last:

@ инициализ. в времена матрице и параллельно  
 матс. по времена

mov r4, #0 → инициализ., оправа  
 mov r5, #0 → матс. суммы в бп.

next-row:

cmp r4, r1 → while i < m  
 bge last-row

@ инициализ. инициализации по индексу  
 брояв и параллельно суммы едем.  
 едем. се обсл. фазы и наименование.

mov r6, #0 → сумма индекса брояв  
 mov r7, #0 → оправа индекса, индекс инициализации  
 ldr r8, [r0, r4, lsl #2] → agree на индексе брояв

next-element:

cmp r7, r2 → while j < n  
 bge last-element

ldr r9, [r8, r7, lsl #2]

cmpl r9, #0

rsblt r9, r9, #0

одинаковые (0-r9) или же r9 неодинаков

add r6, r6, r9

@ инициализ. индекса бп. суммы инициализации  
 инициализации инициализации едем. и увел. #1  
 за бп. едем.

ldr ip, [r10, r7, lsl #2]

add ip, ip, r9

str ip, [r10, r7, lsl #2]

add r7, r7, #1  
b next\_element

last\_element:

@авто же синя то брети beh a og  
go caga тайбета, аттар. нағбын гын

cmp r6, r5  
movgt r5, r6

add r4, r4, #1  
b next\_row

last\_row:

@ оғызыжына маал. синий түс монорана

mov r4, #0  
mov r6, #0 → ишеним маал.

check\_next:

cmp r4, r2  
beq last\_check

ldr r7, [r10, r4, lsl #2]  
cmp r7, r6  
movgt r6, r7

add r4, r4, #1  
b check\_next

last\_check:

slr r5, [r3]  
ldr r3, [sp, #8]  
str r6, [r5]

ldmfd sp!, {r6-r10}

{...}

② qsort.s:

@void qsort (int a[], int l, int r)

указка  
1. elem  
указка  
2. elem

qsort:

{...}

stmfd sp!, {r4-r7}

cmp r1, r2 → r1 > r2  
bge done

mov r4, r1 → r4 = l  
mov r5, r2 → r5 = r  
mov r6, r0

mov r7, r1 → r7 = l

ldr r0, [r6, r4, lsl #2] → r0 жиңбаш  
(күнгілінебі  
енесемін)

next\_element:

cmp r1, r5 < r  
bgt last\_elem

ldr r3, [r6, r1, lsl #2]

cmp r3, r0 } автожиңбаш  
bge continue } бетінан = жиңбаш  
ишина же 4е  
жинобын жиңбаш

@авто же маалу,  
ybetabaxs r7, автожиңбаш  
саналыжыныс r7 урт

r1 > r7  
↓  
11 15 8 2 6  
ro r3

add r7, r7, #1

ldr r2, [r6, r7, lsl #2]  
str r2, [r6, r1, lsl #1]

7. жиңбаш  
6. жиңбаш  
5. жиңбаш  
4. жиңбаш  
3. жиңбаш  
2. жиңбаш  
1. жиңбаш

str r3, [r6, r7, lsl #2]

continue:

add r1, r1, #1 → r1+  
b next\_element

last elem:

@wurabranco suborica ha r7 do.

wurj merles

ldr r3, [r6, r7, lsl #2]

wurjys

str r0, [r6, r7, lsl #2]

baus of impes

str r3, [r6, r9, lsl #2]

fa ogge

mov r0, r1

no v1

mov r1, r4

v2

sub r2, r7, #1

bl qsort

@ qsort(a, r7+1, r)

mov r0, r6

add r1, r7, #1

mov r2, r5

bl qsort

done:

{imfdsp!, {r4-r7}}

{...}