Assembler för INTEL/AMD 64 bitar (x64) del 2

Mål

- Vanliga programkonstruktioner i assembler, t.ex
  - Jämförelse och villkorliga hopp
  - Iteration (while/for)
  - Selektion (if-else)

- Debugging
  - GDB

- Fler kodexempel – med mix av assembler och högnivå
  - Assembler + C
  - C + assembler

Register

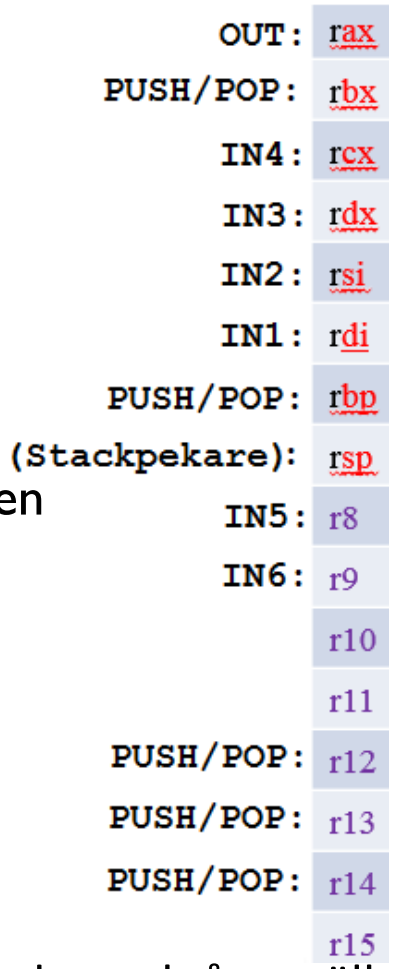
64-bitsregister	32-bitsregister	16-bitsregister	8-bitsregister
rax	eax	ax	al (ah, hög byte i ax)
rbx	ebx	bx	bl (bh, hög byte i bx)
rcx	ecx	cx	cl (ch, hög byte i cx)
rdx	edx	dx	dl (dh, hög byte i dx)
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Addq = 64-bit addition
Addl = 32-bit addition
Addw = 16-bit addition
Addb = 8-bit addition

Samtliga namn (utom specialfallen i parenteser) avser de lägsta bitarna

Anropskonventioner vid subrutin

Gäller AT&T (inte Windows-miljö)



- Heltalsparametrar skickas in (med **`call`**) i **`%rdi`**, **`%rsi`**, **`%rdx`**, **`%rcx`**, **`%r8`**, **`%r9`**
  - första parametern i **`%rdi`**
  - andra parametern i **`%rsi`**
  - **…**
  - Vid fler än 6 parametrar skickas resten via stacken
- Flyttalsparametrar skickas in via **`%xmm`** (se komplett (**`%rax`** innehåller antal **`%xmm`** som används)
- Returvärdet skickas i **`%rax`** som heltal (eller pekare) Vid flyttal: **`%xmm0`**
- **`%rbp`**, **`%rbx`** och **`%r12 – %r14`** måste sparas undan och återställas av en subrutin om deras värden förändras i rutinen

Programmering med subrutiner

Special-instruktioner:

Instruktion	Beskrivning
call *function*	Subrutinanrop till *function (label),* återhoppsadress läggs automatiskt på stacken
ret	Retur från subrutin, återhoppsadress hämtas automatiskt från stacken till pc

- **Stack alignment**
  - Vid anrop av funktion måste stacken vara ***16 bytes aligned***
  - Det innebär att stackpekarens värde måste vara en adress jämnt delbar med 16 (annars blir det ***segmentation fault***)
  - Speciellt viktigt vid anrop till externa funktioner/bibliotek, t.ex. printf() från C standardbibliotek (stdio.h)
  - Ett funktionsanrop lägger återhoppsadressen på stacken (8 bytes) => stacken blir direkt unaligned
    - Inled med *pushq $0* för att få stacken 16 bytes aligned igen
    - eller minska stackpekare direkt med *subq $8, %rsp*
    - Avsluta med *popq* till något register, eller med *subq $-8, %rsp* (eller *addq $8, %rsp*)

Vanliga programkonstruktioner i assembler

- Suffix och datatyper
- Utskrifter med printf (C standarbibliotek)
- Jämförelse och villkorliga hopp
- Iteration (while/for)
- Selektion (if-else)

b, w, l och q suffix

C-kod:

```
int i=0;          /* 32 bit */
long long j=1;    /* 64 bit */
char c='A';       /*  8 bit */
i++;
j++;
c++;
```

INTEL-kod:

```
movl $0, %eax     # 32 bit
movq $1, %rdi     # 64 bit
movb $41, %cl     #  8 bit
incl %eax
incq %rdi
incb %cl
```

64-bit	32-bit	16-bits	8-bit
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl

OUT: rax
PUSH/POP: rbx
IN4: rcx
IN3: rdx
IN2: rsi
IN1: rdi
PUSH/POP: rbp
(Stackpekare): rsp
IN5: r8
IN6: r9
r10
r11
PUSH/POP: r12
PUSH/POP: r13
PUSH/POP: r14
r15

Addq = 64-bit addition
Addl = 32-bit addition
Addw = 16-bit addition
Addb = 8-bit addition

OBS! ”Integer” kan variera med applikation, ofta 32 bitar men ibland 16 bitar.

printf( )

C-kod:

```
int i=10;              /* 32 bit */
printf (”%d \n”, i);
```

INTEL-kod:

```
.data
string: .asciz ”%d \n”
.text
movl $10, %ecx       # 32 bit
leaq string, %rdi    # %rdi pekar på string
                     # Ladda inparameter 1 (ovan)
movl %ecx, %esi      # Ladda inparameter 2
(movq $0, %rax        # Nollställ utparameter)
call printf          # Tänk på stack alignment!
```

64-bit	32-bit	16-bits	8-bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl

OUT:	rax
PUSH/POP:	rbx
IN4:	rcx
IN3:	rdx
IN2:	rsi
IN1:	rdi
PUSH/POP:	rbp
(Stackpekare):	rsp
IN5:	r8
IN6:	r9
	r10
	r11
PUSH/POP:	r12
PUSH/POP:	r13
PUSH/POP:	r14
	r15

while loop

C-kod:

```
while (i<10)
{
//CODE//
}
```

INTEL-kod:

```
whileloop:
    cmpl $10, %ecx    # Beräkna %ecx-10 (eflags)
    jge endwhileloop
    //CODE//
    jmp whileloop
endwhileloop:
```

64-bit	32-bit	16-bits	8-bit
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl

OUT:	rax
PUSH/POP:	rbx
IN4:	rcx
IN3:	rdx
IN2:	rsi
IN1:	rdi
PUSH/POP:	rbp
(Stackpekare):	rsp
IN5:	r8
IN6:	r9
	r10
	r11
PUSH/POP:	r12
PUSH/POP:	r13
PUSH/POP:	r14
	r15

for loop

C-kod:

```
for (int i=0;i<5;i++)
{
//CODE//
}
```

INTEL-kod:

```
    movl $0, %ecx
forloop:
    cmpl $5, %ecx      # Beräkna %ecx-5 (eflags)
    jge endforloop
//CODE//
    incl %ecx
    jmp forloop
endforloop:
```

64-bit	32-bit	16-bits	8-bit
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl

OUT:	rax
PUSH/POP:	rbx
IN4:	rcx
IN3:	rdx
IN2:	rsi
IN1:	rdi
PUSH/POP:	rbp
(Stackpekare):	rsp
IN5:	r8
IN6:	r9
	r10
	r11
PUSH/POP:	r12
PUSH/POP:	r13
PUSH/POP:	r14
	r15

if..else sats

C-kod:

```
if (i==2)
{
//CODE1
}
else
{
//CODE2
}
```

INTEL-kod:

```
    cmpl $2, %ecx          # Beräkna %ecx-2 (eflags)
    jne elsetrue
    //CODE1//
    jmp endif
elsetrue:
    //CODE2//
endif:
```

64-bit	32-bit	16-bits	8-bit
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl

OUT:	rax
PUSH/POP:	rbx
IN4:	rcx
IN3:	rdx
IN2:	rsi
IN1:	rdi
PUSH/POP:	rbp
(Stackpekare):	rsp
IN5:	r8
IN6:	r9
	r10
	r11
PUSH/POP:	r12
PUSH/POP:	r13
PUSH/POP:	r14
	r15

Funktion greater( )

64-bit	32-bit	16-bits	8-bit
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl

OUT: rax
PUSH/POP: rbx
IN4: rcx
IN3: rdx
IN2: rsi
IN1: rdi
PUSH/POP: rbp
(Stackpekare): rsp
IN5: r8
IN6: r9
r10
r11
PUSH/POP: r12
PUSH/POP: r13
PUSH/POP: r14
r15

C-kod:

```
char greater (int *a, int *b)
{
if (*a>*b)
  return 1;
else
  return 0;
}
```

INTEL-kod:

```
greater:
   movl (%rdi), %ecx   # Läs in inparameter 1 från minne till %ecx
   movl (%rsi), %edx   # Läs in inparameter 2 från minne till %edx
   cmpl %ecx, %edx     # Beräkna %edx-%ecx och uppdatera eflags
   jg gtr
   movb $0, %al        # %al = lägsta 8-bitar av %rax
   ret
gtr:
   movb $1, %al        # %al = lägsta 8-bitar av %rax
   ret
```

Debugging med GDB

Se sida i Canvas-modulen ”Laborationer”:

GDB debugtips

Debugging med GDB

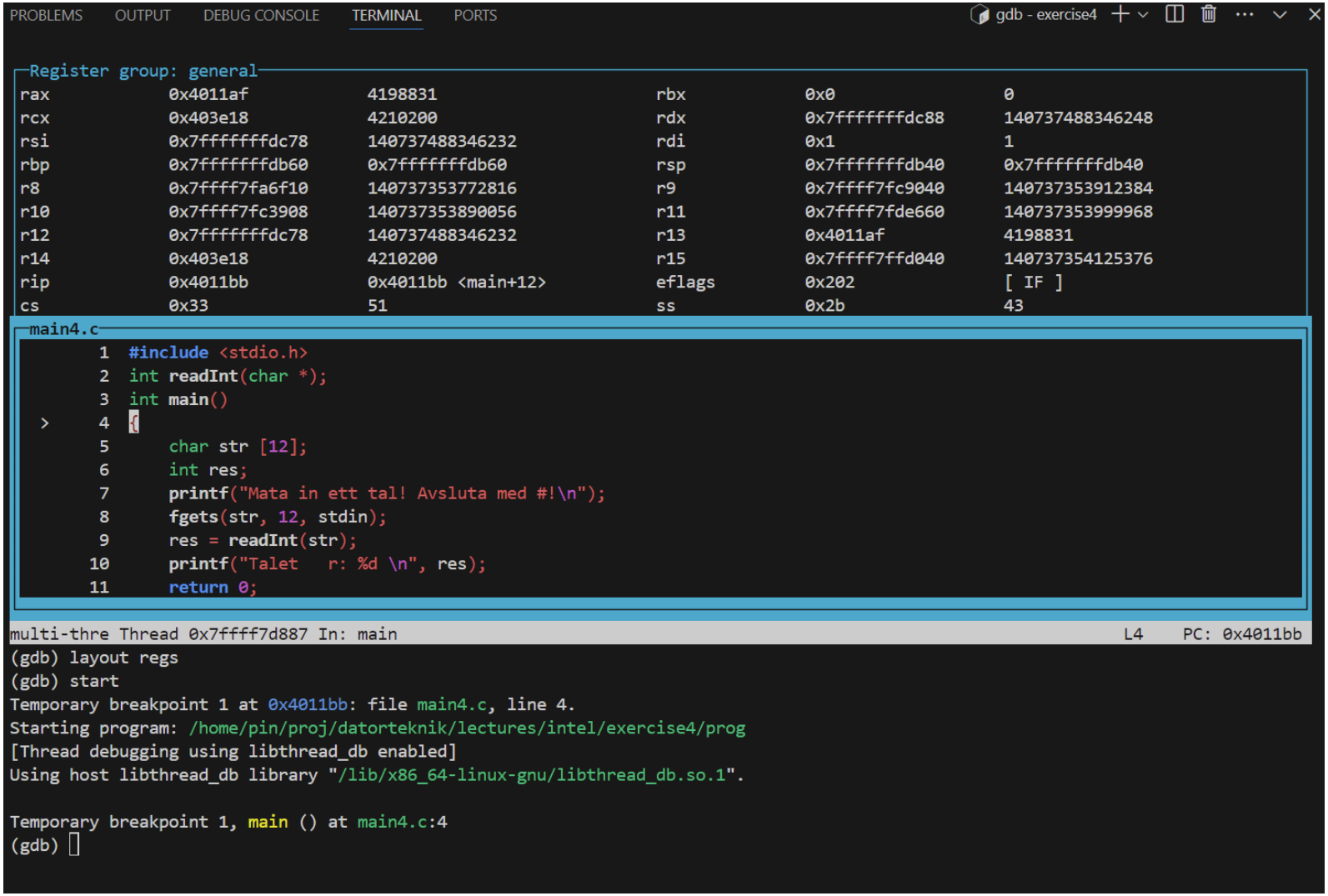
Utdrag ur ***GDB debugtips:***

GDB är en debugger som finns att installera på er Linux-maskin. Det är en textbaserad debugger, men har alla de funktioner man känner igen från andra utvecklingsmiljöer, t.ex. VS Code. Det man saknar är förstås ett grafiskt användargränssnitt, men efter ett tags användning fungerar detta utmärkt.

För generell info om GDB se till exempel:
https://sourceware.org/gdb/current/onlinedocs/gdb.html

Här kommer lite extra tips för hur man kan debugga sitt program (t.ex. assembler- och/eller C-program) med GDB.
*Speciellt kommer vi att använda något som heter **GDB TUI** (**T**ext **U**ser **I**nterface), se mer på*
https://sourceware.org/gdb/current/onlinedocs/gdb.html/TUI.html

Debugging med GDB - exempel



Fler kodexempel

- Fyra uppgifter
- Uppgift 3 och 4 viktiga inför Laboration 3!

- Lösningar i separat presentation och i kod.

Uppgift 1

Skriv en subrutin, max(x,y), som bestämmer vilket av två heltal som är störst där

x = första heltalet
y = andra heltalet
returvärde = det största av de två talen x och y.

Rutinen ska anropas från följande C-program:

```
#include <stdio.h>
int max(int, int);

int main()
{
    int x,y,res;
    printf("Mata in två heltal\n");
    scanf("%d", &x);
    scanf("%d", &y);
    res = max(x,y);
    printf("Talet %d var störst av dem\n", res);
    return 0;
}
```

Noteringar:
x och y = 32-bit integer
x = `%edi`
y = `%esi`
max = `%eax`

OUT:	rax
PUSH/POP:	rbx
IN4:	rcx
IN3:	rdx
IN2:	rsi
IN1:	rdi
PUSH/POP:	rbp
(Stackpekare):	rsp
IN5:	r8
IN6:	r9
	r10
	r11
PUSH/POP:	r12
PUSH/POP:	r13
PUSH/POP:	r14
	r15

Uppgift 2

OUT:	rax
PUSH/POP:	rbx
IN4:	rcx
IN3:	rdx
IN2:	rsi
IN1:	rdi
PUSH/POP:	rbp
(Stackpekare):	rsp
IN5:	r8
IN6:	r9
	r10
	r11
PUSH/POP:	r12
PUSH/POP:	r13
PUSH/POP:	r14
	r15

Antag att vi har en sekvens av n st positiva tal (array). Skriv en subrutin,maxNum, som ger oss det största talet i sekvensen.
Argument 1: Adress till talsekvensen
Argument 2: Antalet tal i sekvensen
Returvärde: Det största av talen i sekvensen

Rutinen ska ingå i ett bibliotek som ska anropas från följande C-progr.:

```
#include <stdio.h>
int maxNum(int *, int);
int main()
{
  int vect[5];
  int res;

  printf("Mata in fem heltal\n");
  scanf("%d", &vect[0]);
  scanf("%d", &vect[1]);
  scanf("%d", &vect[2]);
  scanf("%d", &vect[3]);
  scanf("%d", &vect[4]);
  res = maxNum(vect,5);
  printf("Talet %d var störst av dem\n", res);
  return 0;
}
```

Noteringar:

vect[5] = 5 st 32-bit integer
res = 32-bit integer
(`%rdi`) pekar på vect[0]
`%esi` = 5
maxNum = `%eax`

Uppgift 3

Vi har en teckensträng som startar med siffror avslutas med ”icke-siffra”.
Ex: ”1234X”, ”122Y”, ”13323_” eller ”123123C”

Skriv en rutin, readInt, som går igenom strängen och returnerar ett heltal (det tal vars representation utgörs av siffrorna i strängen).

Argument: Adress till strängen
Returvärde: Utläst heltal
Rutinen ska kunna anropas från följande C-program:

```
#include <stdio.h>
int readInt(char *);

int main()
{
  char str [10];
  int res;
  printf("Mata in ett tal! Avsluta med #!\n");
  fgets(str, 12, stdin);
  res = readInt(str);
  printf("Talet är: %d \n", res);
  return 0;
}
```

Noteringar:
str[10] = 10 st char
res = 32-bit integer
(`%rdi`) pekar på str[0]
readInt = `%eax`

Uppgift 4

Vi har en teckensträng som startar med siffror avslutas med ”icke-siffra”.

Den kan innehålla ett godtyckligt antal blanktecken (noll eller flera) framför talet samt att talet kan inledas med ett tecken ’+’ eller ’-’ före första siffran.

Ex: ” +1234X”, ” -122Y”, ” 13323 ” eller ”123123C”

Modifiera uppgift 3 med den röda inringningen!

Att-göra-lista:

- **Införa en teckenindikator (ifall vi upptäcker minustecken)**
- **Leta upp blanktecken och kasta dem**
- **Leta upp ev. '+'**
- **Leta upp ev. '-'**
- **Om minus: Teckenindikator = 1, annars 0**
- **Kör uppgift 3**
- **Ändra ev. tecken till negativt**

Noteringar:

str[10] = 10 st char
res = 32-bit integer
(`%rdi`) pekar på str[0]
readInt = `%eax`