ARHITECTURA SISTEMELOR DE CALCUL – SEMINAR 5

NOTIȚE SUPORT SEMINAR

Cristian Rusu

ÎNTREBĂRI SCURTE, EX. 1

a) **adresa de memorie cea mai mare accesibilă este**

.

ÎNTREBĂRI SCURTE, EX. 1

a) **adresa de memorie cea mai mare accesibilă este 2^{32} = 4GB (4.294.967.296 bytes)**

b) **instrucțiunea este *jne etichetă*, unde *jne* are opcode 0110**

- adresa de memorie cea mai mare accesibilă este 2^{28}

ÎNTREBĂRI SCURTE, EX. 1

a) **adresa de memorie cea mai mare accesibilă este 2^{32} = 4GB (4.294.967.296 bytes)**

b) **instrucțiunea este *jne etichetă*, unde *jne* are opcode 0110**

- adresa de memorie cea mai mare accesibilă este 2^{28} = 0.25 GB

c) **avem instrucțiunea add R1, R2**

- opcode este 0011

ÎNTREBĂRI SCURTE, EX. 1

a) **adresa de memorie cea mai mare accesibilă este 2^{32} = 4GB (4.294.967.296 bytes)**

b) **instrucțiunea este *jne etichetă*, unde *jne* are opcode 0110**

- adresa de memorie cea mai mare accesibilă este 2^{28} = 0.25 GB

c) **avem instrucțiunea add R1, R2**

- opcode este 0011
- operația suportă putem avea 2^{14} = 16384 regiștri diferiți

d) **avem instrucțiunea add R1, R2, R3**

- opcode este 0100

ÎNTREBĂRI SCURTE, EX. 1

a) **adresa de memorie cea mai mare accesibilă este 2^{32} = 4GB (4.294.967.296 bytes)**

b) **instrucțiunea este *jne etichetă*, unde *jne* are opcode 0110**

- adresa de memorie cea mai mare accesibilă este 2^{28} = 0.25 GB

c) **avem instrucțiunea add R1, R2**

- opcode este 0011
- operația suportă putem avea 2^{14} = 16384 regiștri diferiți

d) **avem instrucțiunea add R1, R2, R3**

- opcode este 0100
- vom avea 9.33 biți pentru fiecare reprezentare a unui registru: două poziții suportă 2^9 = 512 iar o poziție 2^{10} = 1024

COD ASSEMBLY, EX. 2

- **while loop**

```
sum = 0;
i = 0;
while (i < 10) {
        sum = sum + i;
        i = i + 1;
}
```

```
.global main

main:
        ; initializare
        mov $0, i
        mov $0, sum

        ; while loop
et_loop:
        mov sum, %eax
        mov i, %ecx
        add %ecx, %eax
        mov %eax, sum

        inc i

        cmp $10, i
        jne et_loop

        ; afiseaza suma
        mov sum, %eax
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

COD ASSEMBLY, EX. 2

- **while loop**

```
sum = 0;
i = 0;
while (i < 10) {
        sum = sum + i;
        i = i + 1;
}
```

- **rezultatul este?**

```
.global main

main:
        ; initializare
        mov $0, i
        mov $0, sum

        ; while loop
et_loop:
        mov sum, %eax
        mov i, %ecx
        add %ecx, %eax
        mov %eax, sum

        inc i

        cmp $10, i
        jne et_loop

        ; afiseaza suma
        mov sum, %eax
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

COD ASSEMBLY, EX. 2

- **while loop**

```
sum = 0;
i = 0;
while (i < 10) {
        sum = sum + i;
        i = i + 1;
}
```

- **rezultatul este?**
  - 45

```
.global main

main:
        ; initializare
        mov $0, i
        mov $0, sum

        ; while loop
et_loop:
        mov sum, %eax
        mov i, %ecx
        add %ecx, %eax
        mov %eax, sum

        inc i

        cmp $10, i
        jne et_loop

        ; afiseaza suma
        mov sum, %eax
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

FOR LOOP, EX. 3

```
int sum = 0;
int i = 0;
for (i = 0; i < 10; i++)
    sum += i;
```

- **care este diferența între i++ și ++i**
  - este rezultatul același?
  - e o variantă mai eficientă decât cealaltă?

FOR LOOP, EX. 3

```
int sum = 0;
int i = 0;
for (i = 0; i < 10; i++)
    sum += i;
```

- **care este diferența între i++ și ++i**
  - este rezultatul același?
  - e o variantă mai eficientă decât cealaltă?

```
int i = 1;
i++; // == 1 și i == 2
```

```
int i = 1;
++i; // == 2 și i == 2, deci compilatorul nu are nevoie de o variabilă temporară
```

FOR LOOP, EX. 3

- **implementare mai eficientă**
  - mai putină memorie
  - mai puține accesări ale memoriei

```
.global main

main:
        ; initializare
        mov $0, i
        mov $0, sum

        ; while loop
et_loop:
        mov sum, %eax
        mov i, %ecx
        add %ecx, %eax
        mov %eax, sum

        inc i

        cmp $10, i
        jne et_loop

        ; afiseaza suma
        mov sum, %eax
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

FOR LOOP, EX. 3

- **implementare mai eficientă**
  - mai putină memorie
  - mai puține accesări ale memoriei
- **totul în regiștri**
  - ca programul acesta să fie identic cu while
    - la sfârșit
    - mov %eax, sum
    - mov $10, i
- **se poate cu mai puține instrucțiuni?**

```
main:
        ; initializare
        xor %eax, %eax
        xor %ecx, %ecx

        ; while loop
et_loop:
        add %ecx, %eax

        inc %ecx

        cmp $10, %ecx
        jne et_loop

        ; afiseaza suma
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

FOR LOOP, EX. 3

- **implementare mai eficientă**
  - mai putină memorie
  - mai puține accesări ale memoriei
- **totul în regiștri**
  - ca programul acesta să fie identic cu while
    - la sfârșit
    - mov %eax, sum
    - mov $10, i
- **se poate cu mai puține instrucțiuni?**
  - da, parcurgere inversă
- **se poate cu și mai puține instrucțiuni?**

```
main:
        ; initializare
        xor %eax, %eax
        mov $9, %ecx

        ; while loop
et_loop:
        add %ecx, %eax

        dec %ecx
        jnz et_loop

        ; afiseaza suma
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

FOR LOOP, EX. 3

- **implementare mai eficientă**
  - mai putină memorie
  - mai puține accesări ale memoriei
- **totul în regiștri**
  - ca programul acesta să fie identic cu while
    - la sfârșit
    - mov %eax, sum
    - mov $10, i
- **se poate cu mai puține instrucțiuni?**
  - da, parcurgere inversă
- **se poate cu și mai puține instrucțiuni?**
  - da: mov $45, %eax

https://godbolt.org/

```
main:
        ; initializare
        xor %eax, %eax
        mov $9, %ecx

        ; while loop
et_loop:
        add %ecx, %eax

        dec %ecx
        jnz et_loop

        ; afiseaza suma
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

FOR LOOP, EX. 3

- **se poate mai eficient?**
  - loop unrolling

```
int sum = 0;
int i = 0;
for (i = 0; i < 10; i++)
    sum += i;
```

FOR LOOP, EX. 3

- **se poate mai eficient?**
  - loop unrolling

```
int sum = 0;
int i = 0;
for (i = 0; i < 10; i+=2) {
    sum += i;
    sum += i+1;
}
```

  - de ce am vrea să facem așa ceva?

FOR LOOP, EX. 3

- **se poate mai eficient?**
  - loop unrolling

```
int sum = 0;
int i = 0;
for (i = 0; i < 10; i+=2) {
    sum += i;
    sum += i+1;
}
```

  - de ce am vrea să facem așa ceva?
    - mai puține salturi

FOR LOOP, EX. 3

- **se poate mai eficient?**
  - loop unrolling

```
main:
        ; initializare
        xor %eax, %eax
        mov $10, %ecx

        ; while loop
et_loop:
        add %ecx, %eax
        dec %ecx

        add %ecx, %eax
        dec %ecx

        jnz et_loop

        sub $10, %eax

        ; afiseaza suma
        push %eax
        push $formatPrint
        call printf
        pop %ebx
        pop %ebx

        ; flush
        push $0
        call fflush
        pop %ebx

        ; exit
        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

PIPELINE HAZARDS, EX. 4

a) %eax ← %ebx + %ecx

%eax ← %ebx + %edx

b) %ebx ← %ecx + %eax

%eax ← %edx + %eax

c) %eax ← %ebx + %ecx

%edx ← %eax + %edx

d) %eax ← 6

%eax ← 3

%ebx ← %eax + 7

PIPELINE HAZARDS, EX. 4

a) %eax ← %ebx + %ecx

%eax ← %ebx + %edx WAW

b) %ebx ← %ecx + %eax

%eax ← %edx + %eax

c) %eax ← %ebx + %ecx

%edx ← %eax + %edx

d) %eax ← 6

%eax ← 3

%ebx ← %eax + 7

PIPELINE HAZARDS, EX. 4

a) %eax ← %ebx + %ecx

%eax ← %ebx + %edx WAW

b) %ebx ← %ecx + %eax

%eax ← %edx + %eax WAR

c) %eax ← %ebx + %ecx

%edx ← %eax + %edx

d) %eax ← 6

%eax ← 3

%ebx ← %eax + 7

PIPELINE HAZARDS, EX. 4

a) %eax ← %ebx + %ecx

%eax ← %ebx + %edx WAW

b) %ebx ← %ecx + %eax

%eax ← %edx + %eax WAR

c) %eax ← %ebx + %ecx

%edx ← %eax + %edx RAW

d) %eax ← 6

%eax ← 3

%ebx ← %eax + 7

PIPELINE HAZARDS, EX. 4

a) %eax ← %ebx + %ecx

%eax ← %ebx + %edx WAW

b) %ebx ← %ecx + %eax

%eax ← %edx + %eax WAR

c) %eax ← %ebx + %ecx

%edx ← %eax + %edx RAW

d) %eax ← 6

%eax ← 3

%ebx ← %eax + 7 WAW, rezultatul poate fi 10 sau 13

BRANCH PREDICTION, EX. 6

- **ce face algoritmul?**

```
while (na > 0 && nb > 0)
{
    if (*A++ <= *B++) {
        *C++ = *A++; --na;
    } else {
        *C++ = *B++; --nb;
    }
}

while (na > 0) {
    *C++ = *A++; --na;
}

while (nb > 0) {
    *C++ = *B++; --nb;
}
```

BRANCH PREDICTION, EX. 6

- **ce face algoritmul?**
  - merge sort
- **câte instrucțiuni de salt avem?**

```
while (na > 0 && nb > 0)
{
    if (*A++ <= *B++) {
        *C++ = *A++; --na;
    } else {
        *C++ = *B++; --nb;
    }
}

while (na > 0) {
    *C++ = *A++; --na;
}

while (nb > 0) {
    *C++ = *B++; --nb;
}
```

BRANCH PREDICTION, EX. 6

- **ce face algoritmul?**
  - merge sort
- **câte instrucțiuni de salt avem?**
  - 4

```
while (na > 0 && nb > 0)
{
    if (*A++ <= *B++) {
        *C++ = *A++; --na;
    } else {
        *C++ = *B++; --nb;
    }
}

while (na > 0) {
    *C++ = *A++; --na;
}

while (nb > 0) {
    *C++ = *B++; --nb;
}
```

1

2

3

4

BRANCH PREDICTION, EX. 6

- **ce face algoritmul?**
  - merge sort
- **câte instrucțiuni de salt avem?**
  - 4
- **predicția pentru fiecare?**

```
while (na > 0 && nb > 0)
{
    if (*A++ <= *B++) {
        *C++ = *A++; --na;
    } else {
        *C++ = *B++; --nb;
    }
}

while (na > 0) {
    *C++ = *A++; --na;
}

while (nb > 0) {
    *C++ = *B++; --nb;
}
```

1

2

3

4

BRANCH PREDICTION, EX. 6

- **ce face algoritmul?**
  - merge sort
- **câte instrucțiuni de salt avem?**
  - 4
- **predicția pentru fiecare?**
  - Salt 1: sare mereu
  - Salt 2: în general, nu știm
  - Salt 3: sare mereu
  - Salt 4: sare mereu

```
while (na > 0 && nb > 0)
{
    if (*A++ <= *B++) {
        *C++ = *A++; --na;
    } else {
        *C++ = *B++; --nb;
    }
}

while (na > 0) {
    *C++ = *A++; --na;
}

while (nb > 0) {
    *C++ = *B++; --nb;
}
```

1 2 3 4

BRANCH PREDICTION, EX. 6

- **ce face algoritmul?**
  - merge sort
- **câte instrucțiuni de salt avem?**
  - 4
- **predicția pentru fiecare?**
  - Salt 1: sare mereu
  - Salt 2: în general, nu știm
  - Salt 3: sare mereu
  - Salt 4: sare mereu
- **cum eliminăm Saltul 2?**
  - int cmp = (*A <= *B)
  - int min = *B ^ ((*B ^ *A) & (-cmp))
    - *C++ = min
    - A += cmp, na -= cmp
    - B += !cmp, nb -= !cmp

```
while (na > 0 && nb > 0)
{
    if (*A++ <= *B++) {
        *C++ = *A++; --na;
    } else {
        *C++ = *B++; --nb;
    }
}

while (na > 0) {
    *C++ = *A++; --na;
}

while (nb > 0) {
    *C++ = *B++; --nb;
}
```

1 2 3 4

COD ASSEMBLY. EX. 7

```
        .globl  f
f:
        movl    $1, %r8d
        jmp     .LBB0_1
.LBB0_6:
        incl    %r8d
.LBB0_1:
        movl    %r8d, %ecx
        imull   %ecx, %ecx
        movl    $1, %edx
.LBB0_2:
        movl    %edx, %edi
        imull   %edi, %edi
        movl    $1, %esi
        .align  16, 0x90
.LBB0_3:
        movl    %esi, %eax
        imull   %eax, %eax
        addl    %edi, %eax
        cmpl    %ecx, %eax
        je      .LBB0_7
        cmpl    %edx, %esi
        leal    1(%rsi), %eax
        movl    %eax, %esi
        jl      .LBB0_3
        cmpl    %r8d, %edx
        leal    1(%rdx), %eax
        movl    %eax, %edx
        jl      .LBB0_2
        jmp     .LBB0_6
.LBB0_7:
        pushq   %rax
.Ltmp0:
        movl    $.L.str, %edi
        xorl    %eax, %eax
        callq   printf
        movl    $1, %eax
        popq    %rcx
        retq

.L.str:
        .asciz  "%d %d\n"
        .size   .L.str, 7
```

.

COD ASSEMBLY. EX. 7

```
        .globl  f
f:
        movl    $1, %r8d
        jmp     .LBB0_1
.LBB0_6:
        incl    %r8d
.LBB0_1:
        movl    %r8d, %ecx
        imull   %ecx, %ecx
        movl    $1, %edx
.LBB0_2:
        movl    %edx, %edi
        imull   %edi, %edi
        movl    $1, %esi
        .align  16, 0x90
.LBB0_3:
        movl    %esi, %eax
        imull   %eax, %eax
        addl    %edi, %eax
        cmpl    %ecx, %eax
        je      .LBB0_7
        cmpl    %edx, %esi
        leal    1(%rsi), %eax
        movl    %eax, %esi
        jl      .LBB0_3
        cmpl    %r8d, %edx
        leal    1(%rdx), %eax
        movl    %eax, %edx
        jl      .LBB0_2
        jmp     .LBB0_6
.LBB0_7:
        pushq   %rax
.Ltmp0:
        movl    $.L.str, %edi
        xorl    %eax, %eax
        callq   printf
        movl    $1, %eax
        popq    %rcx
        retq

.L.str:
        .asciz  "%d %d\n"
        .size   .L.str, 7
```

verifică $x^2 + y^2 = z^2$, cu condiția $x \leq y$

.