Arhitectura Sistemelor de Calcul Decembrie, 2021

Seminar 0x04

Cristian Rusu

1 Scopul seminarului

În acest seminar vom rezolva niște probleme care implică:

- cod Assembly x86;
- seturi de instrucțiuni;
- pipelining, branch prediction, out of order execution.

2 Exerciții

1. Presupunem ca avem un sistem de calcul pe 32 de biți, răspundeți la următoarele întrebări scurte:
   (a) care este adresa de memorie cea mai mare care poate fi accesată? (cu 8 biți / locație memorie)
   (b) avem instrucțiunea *jne etichetă*, unde *jne* are opcode-ul 0110. Care este saltul maxim care se poate realiza cu această instrucțiune?
   (c) avem o instrucțiune *add R1, R2*, unde *add* are opcode-ul 0011 iar *R1* și *R2* sunt regiștri iar calculul realizat este *R2* ← *R1 + R2*. Câți regiștri diferiți putem avea?
   (d) similar cu instrucțiunea anterioară, dar acum avem *add R1, R2, R3*, unde add are opcode-ul 0100 iar calculul realizat este *R3* ← *R1 + R2*. Câți regiștri diferiți putem avea?

2. Scrieți secvențe scurte de cod Assembly x86 (și verificați pe https://godbolt.org/) pentru a descrie următoarele secvențe de cod C (*x* este un array de 20 valori int iar celelalte variable sunt int):
   (a) $a = 42$
   (b) $b = 10 \times c + 13$
   (c) $y = (a - 5) \times (y + 1337)$
   (d) if $(a == 0)$ $b = b + 1$ else $b = b - 1$
   (e) if $(a == 3)$ $b = b \times 2$ else $b = b/2$
   (f) $x[i] = x[i - 1]$
   (g) $z = x[5] + x[10]$
   (h) if $(a == 0 \ || \ a <= 50)$ $c = 0$ else $c = 1$
   (i) while (1) $\{ \ldots \}$
   (j) $\text{sum} = 0;\ i = 0;\ \text{while}(i < 10)\ \{\ \text{sum} = \text{sum} + i;\ i = i + 1;\}$

   a)
   ```
   a:
       .long   42
   ```

   b)
   ```
   c:
       .long   19
   b:
       .long   203
   ```

   c)
   ```
   mov    eax, dword ptr [rbp - 8]
   sub    eax, 5
   mov    ecx, dword ptr [rbp - 12]
   add    ecx, 1337
   imul   eax, ecx
   mov    dword ptr [rbp - 12], eax
   ```

3. Fie următoarea secvență de cod scrisă în C:

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

   Cerințe:
   (a) în ciclul for de mai sus putem folosi i++ sau ++i. Există o diferență între cele două operații?

(b) scrieți cod Assembly echivalent cu această secvență;

(c) comparați rezultatul cu secvența de cod de la exercițiul predecent, subpunctul (i). Care variantă este mai eficientă?

(d) rescrieți secvența de C de mai sus dar de data aceasta la fiecare iterație actualizați suma de două ori (acum suma este actualizată o singură dată).

4. În contextul pipelining, când codul sursă primit are dependențe de date, ce fel de erori (hazards) reprezintă următoarele secvențe?

   (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

5. Aveți un calculator al cărui CPU are două unități principale: 1) o unitate care încarcă date din memorie în regiștri (câte o variabilă o dată) și 2) o unitate aritmetică/logică care poate executa două instrucțiuni simultan. Calculați cât mai eficient pe această mașină expresia $a + b + a \times c + b \times c + d + d \times e$.

6. Considerăm următoarea secvență de cod C unde A și B sunt vectori iar na și nb reprezintă dimensiunile celor doi vectori:

```
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;
}
```

   Cerințe:

   (a) ce face algoritmul de mai sus?

   (b) câte instrucțiuni de branch (salt) există în codul de mai sus?

   (c) puteți prezice eficient pentru fiecare branch dacă acesta sare sau nu?

   (d) dacă au fost salturi pe care nu le puteți prezice, schimbați codul de mai sus astfel încât să le eliminați.

7. Scrieți o funcție toUpper() care ia ca parametru un șir de caractere și returnează același șir dar în care toate caracterele sunt majuscule (limbajul de programare nu este important, puteți scrie pseudo-cod). Scrieți inițial o soluție cu salturi și apoi încercați să eliminați salturile complet. Gândiți-vă cum puteți îmbunătății soluția și mai mult.

2. d)

```
if (a == 0)
    b = b + 1;
else
    b = b - 1;
```

```
    cmp     dword ptr [rbp - 8], 0
    jne     .LBB1_2
    mov     eax, dword ptr [rbp - 12]
    add     eax, 1
    mov     dword ptr [rbp - 12], eax
    jmp     .LBB1_3
.LBB1_2:
    mov     eax, dword ptr [rbp - 12]
    sub     eax, 1
    mov     dword ptr [rbp - 12], eax
```

e)

```
if (a == 3)
    b = b * 2;
else
    b = b / 2;
```

```
    cmp     dword ptr [rbp - 8], 3
    jne     .LBB1_2
    mov     eax, dword ptr [rbp - 12]
    shl     eax, 1
    mov     dword ptr [rbp - 12], eax
    jmp     .LBB1_3
.LBB1_2:
    mov     eax, dword ptr [rbp - 12]
    mov     ecx, 2
    cdq
    idiv    ecx
    mov     dword ptr [rbp - 12], eax
```

f)

```
int x[100];
int i = 5;
x[i] = x[i - 1];
```

```
    mov     dword ptr [rbp - 420], 5
    mov     eax, dword ptr [rbp - 420]
    sub     eax, 1
    cdqe
    mov     ecx, dword ptr [rbp + 4*rax - 416]
    movsxd  rax, dword ptr [rbp - 420]
    mov     dword ptr [rbp + 4*rax - 416], ecx
```

g)

```
int x[100] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, z;
z = x[5] + x[10];
```

```
    cmp     dword ptr [rbp - 8], 0
    jne     .LBB1_2
    mov     eax, dword ptr [rbp - 12]
    add     eax, 1
    mov     dword ptr [rbp - 12], eax
    jmp     .LBB1_3
.LBB1_2:
    mov     eax, dword ptr [rbp - 12]
    sub     eax, 1
    mov     dword ptr [rbp - 12], eax
```

h)

```
int a = 5, c;
if (a == 0 || a <= 50)
    c = 0;
else
    c = 1;
```

```
    mov     dword ptr [rbp - 8], 5
    cmp     dword ptr [rbp - 8], 0
    je      .LBB1_2
    cmp     dword ptr [rbp - 8], 50
    jg      .LBB1_3
.LBB1_2:
    mov     dword ptr [rbp - 12], 0
    jmp     .LBB1_4
.LBB1_3:
    mov     dword ptr [rbp - 12], 1
```

i)

```
int a = 5;
while (1){
    a = 5;
}
```

```
        mov     dword ptr [rbp - 8], 5
.LBB1_1:                                # =>This Inner Loop Header: Depth=1
        mov     dword ptr [rbp - 8], 5
        jmp     .LBB1_1
```

j)

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

```
        mov     dword ptr [rbp - 8], 0
        mov     dword ptr [rbp - 12], 0
.LBB1_1:                                # =>This In
        cmp     dword ptr [rbp - 12], 10
        jge     .LBB1_3
        mov     eax, dword ptr [rbp - 8]
        add     eax, dword ptr [rbp - 12]
        mov     dword ptr [rbp - 8], eax
        mov     eax, dword ptr [rbp - 12]
        add     eax, 1
        mov     dword ptr [rbp - 12], eax
        jmp     .LBB1_1
```

Instruction	Meaning
CDQE	Sign extend EAX into RAX
CQO	Sign extend RAX into RDX:RAX
CMPSQ	CoMPare String Quadword
CMPXCHG16B	CoMPare and eXCHanGe 16 Bytes
IRETQ	64-bit Return from Interrupt
JRCXZ	Jump if RCX is zero
LODSQ	LoaD String Quadword
MOVSXD	MOV with Sign Extend 32-bit to 64-bit
POPFQ	POP RFLAGS Register
PUSHFQ	PUSH RFLAGS Register
RDTSCP	ReaD Time Stamp Counter and Processor ID
SCASQ	SCAn String Quadword
STOSQ	STOre String Quadword
SWAPGS	Exchange GS base with KernelGSBase MSR