INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ORGANIZAÇÃO DE COMPUTADORES

LEIC

Conjunto de Exercícios IV

Execução de Programas

Versão 3.0

Soluções

2023/2024

Exercício 1

Considere um processador de 32 bits cuja arquitetura lógica (*instruction set architecture - ISA*) tem as seguintes características:

- existem 16 registos de uso geral, R0 - R15;
- as instruções aritméticas e lógicas só operam com registos
  $Rx \leftarrow Ry\ op\ Rz$
  e atualizam a *flag* Zero;
- existem só dois modos de endereçamento
  - indireto por registo: $Rx \leftarrow M[Ry]$
    em que $M[Ry]$ representa o conteúdo da posição de memória cujo endereço está armazenado em Ry;
  - imediato: $Rx \leftarrow W$
    em que W é uma constante;
- a memória é endereçável ao *byte*;
- todas as instruções são codificadas em 32 bits e estão bem alinhadas em memória; os dados com 32 bits (ex. inteiros) também estão bem alinhados;

Neste computador foi compilado o programa em C:

```
#define SIZE 100
int a[SIZE]; /* inteiros com 32 bits */
int i, sum;
sum = 0;
for (i=0; i<SIZE; i++)
{
    sum = sum + a[i];
}
```

que gerou a seguinte sequência de instruções. (Para não entrar no detalhe das mnemónicas e da formatação do *assembly* as instruções são representadas por transferências de registos, a instrução de salto é auto-explicativa.)

```
        R1 ← 0
        R3 ← BASE_A
        R4 ← 4
        ; calcula limite do vetor
        R5 ← SIZE_A ; SIZE_A = SIZE * 4
        R5 ← R3 + R5 ; limite = base + size
Ciclo:
        R2 ← M[R3]
        R1 ← R1 + R2 ; soma
        R3 ← R3 + R4 ; avança no vetor
        R6 ← R5 - R3
        JumpIfNotZero Ciclo
```

(a) Identifique as variáveis e as constantes declaradas no programa fonte. Onde é que o compilador aloca espaço para estas variáveis e constantes?

(b) O que representa $BASE\_A$?

(c) Onde é armazenado o resultado da soma?

(d) Qual é a dimensão da memória alocada pelo compilador para o código, os dados, a pilha e o *heap*?

(e) Qual é a frequência relativa dos tipos de instruções (aritméticas/lógicas, transferências de dados, controlo) no **código gerado**? (Distribuição estática de instruções por tipos.)

(f) Indique o fluxo de acessos entre o processador e a memória (o rasto de execução) correspondente à execução das primeiras 16 instruções do programa. Especifique em cada acesso:
i. o tipo - F (instruction fetch), R - (read data), W (write data), e
ii. o endereço acedido - 0x0000kkkk.
(Considere que o compilador alocou o código a partir de 0x00000000 e os dados a partir de 0x00008000.)

(g) Qual é a frequência relativa dos tipos de instruções (aritméticas/lógicas, transferências de dados, controlo) no **código executado**? (Distribuição dinâmica de instruções por tipos.)

(h) Considerando que as instruções têm tempo de execução constante, e independente do seu tipo - de 20ns, quanto tempo demora aproximadamente executar a soma dos elementos de um vetor de tamanho SIZE?

(i) Está-se a avaliar a modificação do microprocessador de modo a permitir realizar operações aritméticas e lógicas com um operando em memória

$$Rx \leftarrow Ry\ op\ M[Rz]$$

Todavia o aumento de complexidade daqui resultante vai certamente aumentar o tempo de execução das instruções (que se mantém uniforme, independentemente do tipo). A partir do programa em análise, determine o valor máximo do novo tempo de execução das instruções de modo a não penalizar o desempenho.

Exercício 2

Considere um processador de 32 bits cujo conjunto de instruções tem as seguintes características:

- existem 32 registos de uso geral, R0 - R31;
- as instruções aritméticas e lógicas só operam com registos
  $Rx \leftarrow Ry\ op\ Rz$
- o processador não tem flags
- existem só dois modos de endereçamento
  - indireto por registo: $Rx \leftarrow M[Ry]$ou $M[Ry] \leftarrow Rx$, em que $M[Ry]$ representa o conteúdo da posição de memória cujo endereço está armazenado em Ry; esta é a única forma de transferir dados entre memória primária e os registos
  - imediato: $Rx \leftarrow W$, em que W é uma constante
- todas as instruções são codificadas em 32 bits e estão bem alinhadas em memória;
- o espaço de endereçamento tem 2^{32} octetos, sendo endereçáveis octetos e palavras de 32 bits; o formato das palavras é little endian.

O programa seguinte em C calcula um vetor soma, em que cada elemento, $sum[k]$, armazena a soma dos primeiros k+1 elementos de um vetor de entrada, a[].

```
#define SIZE 100
int a[SIZE], sum[SIZE]; /* inteiros com 32 bits */
register int i;
sum[0] = a[0];
for (i=1; i<SIZE; i++)
{
    sum[i] = sum[i-1] + a[i];
}
```

A compilação deste programa gerou a seguinte sequência de instruções. (Para não entrar no detalhe das mnemónicas e da formatação do *assembly* as instruções são representadas por transferências de registos, a instrução de salto é auto-explicativa.)

```
        R6 ← 4
        R3 ← BASE_A
        R4 ← BASE_SUM
        ; calcula limite do vetor
        R5 ← SIZE_A ; SIZE_A = SIZE * 4
        R5 ← R3 + R5 ; BASE_A + SIZE_A (=BASE_SUM)
        R5 ← R5 - R6 ; endereço do último elemento de a[]
        ; calcula vetor soma
        R8 ← M[R3] ; le a[0]
        M[R4] ← R8 ; escreve sum[0]
Ciclo:
        ; sum[i] = sum[i-1] + a[i]
        R3 ← R3 + R6
        R8 ← M[R3] ; le a[i]
        R9 ← M[R4] ; le sum[i-1]
        R10 ← R8 + R9
        R4 ← R4 + R6
        M[R4] ← R10 ; sum[i]
        ; verificação de fim do ciclo
        R7 ← R5 - R3
        JumpIfNotZero R7, Ciclo
```

(a) Assumindo que o compilador alocou o código gerado a partir do endereço 0x00000000 e os dados a partir do endereço 0x00008000, sendo as variáveis alocadas pela ordem em que são declaradas no programa fonte, indique o mapa de ocupação de memória resultante.

(b) Quantos acessos à memória são gerados ao executar o programa? Qualifique os acessos em função da sua natureza: leitura de instruções (F – fetch), leitura de dados (R – read) e escrita de dados (W – write).

(c) Imediatamente antes da execução do programa, verifica-se que a memória tem os conteúdos indicados na tabela seguinte:

Endereço	Conteúdo
0x00008000	0Dh
0x00008001	00h
0x00008002	00h
0x00008003	00h
0x00008004	F2h
0x00008005	FFh
0x00008006	FFh
0x00008007	FFh
0x00008008	14h
0x00008009	00h
0x0000800A	00h
0x0000800B	00h

Qual seria o conteúdo destas posições de memória se o formato das palavras fosse *big endian*? Indique os valores de sum[0], sum[1] e sum[2] calculados pelo programa. Apresente os resultados na base decimal e em hexadecimal, nesta em complemento para 2.

(d) O tempo de execução de cada instrução é $p * 10ns + N * 20ns$ em que p é o número de instruções que fazem processamento interno e N é o número de acessos a memória necessários para processar (ler e executar) a instrução.
Qual é o tempo de execução deste programa?
Qual é o desempenho do processador, em instruções/segundo, ao executar este programa?

(e) O compilador que gerou o código acima é algo "primitivo". Sugira uma otimização no código gerado que, sem alterar a funcionalidade, melhore o desempenho.
Indique o speedup obtido face à compilação inicial.
Qual é, agora, o desempenho, medido em instruções/segundo (IPS)?

(f) Está-se a avaliar a modificação de microprocessador de modo a permitir realizar operações aritméticas e lógicas com um operando em memória, $Rx \leftarrow Ry\ op\ M[Rz]$. Todavia o aumento de complexidade daqui resultante vai certamente aumentar o tempo de processamento interno das intruções - p - que se mantém uniforme. A partir do programa em análise, determine o valor máximo do novo tempo de processamento interno das instruções de modo a não penalizar o desempenho.

Conjunto de Exercícios IV
Arquitectura Lógica

Soluções

Exercício 1

(a) As variáveis escalares são alocadas em registos: i em R3, sum em R1.
O vetor a[] é alocado em memória.
A constante SIZE_A é calculada pelo compilador e embebida no código da instrução:
$R5 \leftarrow$ SIZE_A

(b) BASE_A representa o endereço da base do vetor, isto é, o endereço de a[0].

(c) Em R1

(d) Código: 40B. Dados: 400B. Não é alocada memória para a pilha nem para o *heap*.

(e) Aritméticas/lógicas: 8/10 = 80%
Transferência de dados: 1/10 = 10%
Controlo: 1/10 = 10%

(f)

Instrução	Tipo de Acesso	Endereço
R1 ← 0	F	0x0000
R3 ← BASE_A	F	0x0004
R4 ← 4	F	0x0008
R5 ← SIZE_A	F	0x000C
R5 ← R3 + R5	F	0x0010
Ciclo: R2 ← M[R3]	F,R	0x0014,0x8000
R1 ← R1 + R2	F	0x0018
R3 ← R3 + R4	F	0x001C
R6 ← R5 - R3	F	0x0020
JumpIfNotZero Ciclo	F	0x0024
Ciclo: R2 ← M[R3]	F,R	0x0014,0x8004
R1 ← R1 + R2	F	0x0018
R3 ← R3 + R4	F	0x001C
R6 ← R5 - R3	F	0x0020
JumpIfNotZero Ciclo	F	0x0024
Ciclo: R2 ← M[R3]	F,R	0x0014,0x8008

(g) Aritméticas/lógicas: 3/5 = 60%
Transferência de dados: 1/5 = 20%
Controlo: 1/5 = 20%

(h) 10 000ns

(i) 25ns

Exercício 2

(a) O código ocupa do endereço 0x00000000 até ao 0x0000003F.

Os dados ocupam do endereço 0x00008000 ao 0x0000818F para o vetor a[] e do endereço 0x00008190 ao 0x0000831F para o vetor sum[].

(b) Fetch: $8 + 8 * 99 = 800$
Read: $1 + 2 * 99 = 199$
Write: $1 + 1 * 99 = 100$
Total: $800 + 199 + 100 = 1099$

(c)

Endereço	Conteúdo	Conteúdo "big endian"
0x00008000	0Dh	00h
0x00008001	00h	00h
0x00008002	00h	00h
0x00008003	00h	0Dh
0x00008004	F2h	FFh
0x00008005	FFh	FFh
0x00008006	FFh	FFh
0x00008007	FFh	F2h
0x00008008	14h	00h
0x00008009	00h	00h
0x0000800A	00h	00h
0x0000800B	00h	14h

$sum[0] = 0000000Dh = 13_{10}$;
$sum[1] = FFFFFFFFh = -1_{10}$;
$sum[2] = 00000013h = 19_{10}$

(d) Tempo de execução: 29 980 ns
Desempenho: 26 684 456,30 IPS $\approx$ 26,68 MIPS (milhões de instruções por segundo)

(e)

```
        R6 ← 4
        R3 ← BASE_A
        R4 ← BASE_SUM
        ; calcula limite do vetor
        R5 ← SIZE_A ; SIZE_A = SIZE * 4
        R5 ← R3 + R5 ; limite = base + size - 1
        R5 ← R5 - R6
        ; calcula vetor soma
        R8 R10 ← M[R3] ; le a[0]
        M[R4] ← R8 R10 ; escreve sum[0]
Ciclo:
        ; sum[i] = sum[i-1] + a[i]
        R3 ← R3 + R6
        R8 ← M[R3] ; le a[i]
        R9 ← M[R4] ; le sum[i-1]
        R10 ← R8 + R9 R10
        R4 ← R4 + R6
        M[R4] ← R10 ; sum[i]
        ; verificação de fim do ciclo
        R7 ← R5 - R3
```

```
        JumpIfNotZero R7, Ciclo
```

Speedup$_{executionTime} \approx 1,2$

Desempenho: 28 006 392,33 IPS $\approx$ 28,01 MIPS

(f)
```
Ciclo:
        ; sum[i] = sum[i-1] + a[i]
        R3 ← R3 + R6
        R8 ← M[R3] ; le a[i]
        R9 ← M[R4] ; le sum[i-1]
        R10 ← R8 + R9
        R10 ← R8 + M[R4]
        R4 ← R4 + R6
        M[R4] ← R10 ; sum[i]
        ; verificação de fim do ciclo
        R7 ← R5 - R3
        JumpIfNotZero R7, Ciclo
```

p$_{max}$ = 14,24ns