

Prova on line 2 | e-Disciplinas

Questão 1

Resposta salva

Vale 5,00 ponto(s).

Texto da questão

Considere o programa:

```
#include
```

```
int main () {  
    float ouro = 1.61803398874989484820458683436563811; // https://pt.wikipedia.org/wiki/Lista\_de\_  
    printf ("%0.19f\n", ouro);  
}
```

O código assembly gerado é:

```
.file "float.c"  
.text  
.section .rodata  
.LC1:  
.string "%0.19f\n"  
.text  
.globl main  
.type main, @function  
main:  
.LFB0:  
.cfi_startproc  
endbr64  
pushq %rbp  
.cfi_def_cfa_offset 16  
.cfi_offset 6, -16  
movq %rsp, %rbp  
.cfi_def_cfa_register 6  
subq $16, %rsp  
movss .LC0(%rip), %xmm0  
movss %xmm0, -4(%rbp)  
cvtss2sd -4(%rbp), %xmm0  
leaq .LC1(%rip), %rdi  
movl $1, %eax  
call printf@PLT  
movl $0, %eax  
leave  
.cfi_def_cfa 7, 8  
ret  
.cfi_endproc  
.LFE0:  
.size main, .-main  
.section .rodata  
.align 4  
.LC0:  
.long 1070537661
```

```

    .ident    "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0"
    .section   .note.GNU-stack,"",@progbits
    .section   .note.gnu.property,"a"
    .align 8
    .long      1f - 0f
    .long      4f - 1f
    .long      5
0:
    .string    "GNU"
1:
    .align 8
    .long      0xc0000002
    .long      3f - 2f
2:
    .long      0x3
3:
    .align 8
4:

```

a-) Onde, no código em assembly, está a constante 1.61803398874989484820458683436563811 (use os rótulos como referência) - 1,5pt

b-) Mostre, **por conversão dos números, indique as contas, apresente resultados intermediários**, que a constante que você identificou no assembly corresponde a 1.61803398874989484820458683436563811 - 2pt

c-) Há erros de arredondamento que iniciam por volta da sexta casa decimal, por quê? - 1,5pt

Texto de resposta Questão 1

Editor de texto rico

a) Os dados em assembly são armazenados dentro da seção ".rodata"(ready-only data), em específico, as constantes ficam armazenadas no rótulo ".LCx", que o "GCC" tipicamente cria para armazenar dados locais. Nesse caso, como foi usado um *float*, o compilador guarda um decimal de 32 bits para representar o flutuante especificado por um número flutuante de precisão única pela IEEE 754. Assim o número está representado em ".long 1070537661".

b) Para se fazer a conversão do decimal guardado em memória para a representação flutuante, pela norma IEEE 754, deve-se converter o número "1070537661" em base 10 para binário, que então fica: "001111111001111000110111011101". Como é um flutuante de precisão única, há 1 bit de sinal, 8 bits de expoente viesado em 127 e 23 bits de mantissa, separando respectivamente tem-se: "0", "01111111" e "1001111000110111011101". Dessa forma, como o bit de sinal é 0, é um número positivo. Convertendo o expoente para decimal tem-se 127 que desviesando tem-se $127 - 127 = 0$, logo o expoente é 0. Para a mantissa, é necessário converter de forma fracionada para decimal, assim tem-se "0.1001111000110111011101", que para decimal resulta em aproximadamente "0.6180340051651000977". Para achar o número flutuante usa-se a fórmula; $(-1)^{\text{bit de sinal}} * 2^{\text{expoente}} * 1.\text{mantissa}$. Portanto, o número armazenado no assembly é $(-1)^0 * 2^0 * 1.6180340051651000977 = 1.6180340051651000977$.

c) Em ponto flutuante de precisão única há apenas 23 bits de mantissa, dessa maneira, não há como representar infinitos números reais, sendo necessário arredondar em certa precisão. Para calcular quando há necessidade de arredondar, é possível usar a fórmula $\log_{10}(2^{\text{bits de mantissa}})$, que para a precisão usada, ficaria $\log_{10}(2^{23})$ que é aproximadamente 6,92 casas decimais. Assim, por volta de 6 casas há a necessidade de usar alguma técnica de arredondamento já convencionada pela norma IEEE 754.

Questão 2

Resposta salva

Vale 5,00 ponto(s).

Texto da questão

Considere dois sistemas, ambos com memória de 16MBytes, um com cache de endereçamento direto de 16KLinhas e outro com cache de endereçamento associativo por conjunto em duas vias também com 16KLinhas. Uma linha tem 4 palavras, a palavra tem 1 Byte.

Em ambos sistemas, o processador acessa os endereços que contém dados no diagrama, em sequência crescente de endereços. Ié, acessa os endereços 0x000000, 0x000001, 0x000002, 0x000003, em seguida, 0x160000, 0x160001, 0x160002, 0x160003, em seguida, 0x0x160004, 0x160005, 0x160006, 0x160007, em seguida, 0x16339C, 0x16339D, 0x16339E, 0x16339F, e assim por diante.

a-) Nessa sequência de operação, quando há substituição de linha do cache no sistema com cache de endereçamento *direto*? Por exemplo, "quando o processador acessar o endereço X, o conteúdo da linha L é substituído". Quantas substituições são feitas? - 1,5pt

b-) Nessa sequência de operação, quando há substituição de linha do cache no sistema com cache de endereçamento *associativo por conjunto*? Por exemplo, "quando o processador acessar o endereço X, o conteúdo da linha L da via V é substituído". Quantas substituições são feitas? - 1,5pt

c-) No sistema com cache de endereçamento associativo por conjunto, após a sequência de acessos ser completada, acrescente um acesso de maneira a forçar uma substituição de linha, diga que linha deve ser substituída usando a política LRU. 2pt.

Como apoio, são apresentados os diagramas 4.10 e 4.15 do livro Computer Organization and Architecture de STALLINGS, W. que ilustram a sequência de acessos à memória e a atualização do cache.

 Diagrama mostrando memória principal e cache de endereçamento direto

Texto de resposta Questão 2

Editor de texto rico

a) Para calcular as substituições de linha, é necessário calcular primeiramente o modo de designar a linha do cache para cada endereçamento da memória primária, para isso, no sistema de cache por endereçamento direto é feita uma função hash da seguinte maneira: " $i = j \bmod m$ ", onde i é o número da linha no cache, j o número do bloco na memória principal, e m o número total de linhas no cache. O m é dado pelo exercício como $16 \cdot 1024$ linhas, o j é calculado a partir da quantidade de palavras que o cache armazena por linha, ou seja, quantos endereços de memória sequenciais o cache guarda em uma linha, nesse caso, o exercício dá como 4 palavras, dessa forma para descobrir o bloco da memória principal, basta pegar o resultado da divisão euclidiana entre o endereço e 4, exemplo: $0x16339C/4 = 0x0CE7$. Assim, a quantidade de dados distintas que uma linha do cache pode receber se dá por " $(\text{tamanho da RAM}) / (\text{palavras por linha} \cdot \text{tamanho do cache}) = (16 \cdot 1024 \cdot 1024) / (4 \cdot 16 \cdot 1024) = 256$."

Para o acesso aos dados do diagrama é dado 7 sequências de 4 palavras cada, assim é feita uma escrita no cache por sequência. Dessa forma calculando a linha do cache de cada sequência tem-se: $i_0 = 0x000000/4 \bmod 16 \cdot 1024 = 0x0$, $i_1 = 0x160000/4 \bmod 16 \cdot 1024 = 0x0$, $i_2 = 0x160004/4 \bmod 16 \cdot 1024 = 0x1$, $i_3 = 0x16339C/4 \bmod 16 \cdot 1024 = 0xCE7$, $i_4 = 0x16FFFC/4 \bmod 16 \cdot 1024 = 0x3FFF$, $i_5 = 0xFFFFF8/4 \bmod 16 \cdot 1024 = 0x3FFE$, $i_6 = 0xFFFFFC/4 \bmod 16 \cdot 1024 = 0x3FFF$. Portanto, é possível visualizar 2 substituições: ao acessar a sequência que se inicia em i_1 , pois essa linha já estava com o dado de i_0 e ao acessar a sequência que se inicia em i_6 , pois já estava ocupada com a sequência que se inicia em i_4 .

b) Para o sistema de cache com endereço associativo por conjunto, o cálculo do hash muda um pouco em relação ao sistema direto, da seguinte maneira: " $i = j \bmod v$ ", o j é o mesmo do endereçamento direto citado em "a", o v é número de conjuntos, que pode ser calculado do seguinte jeito: " $v = m/k$ ", onde m é o número de linhas no cache, e k o número de linhas em cada conjunto. O k é dado no exercício ao afirmar que há duas vias e m é $16 \cdot 1024$, assim $v = 16 \cdot 1024 / 2 = 8192$.

Dessa forma, para o acesso aos dados do diagrama é feito os cálculos: $i_0 = 0x000000/4 \bmod 8192 = 0x0$, $i_1 = 0x160000/4 \bmod 8192 = 0x0$, $i_2 = 0x160004/4 \bmod 8192 = 0x1$, $i_3 = 0x16339C/4 \bmod 8192 = 0xCE7$, $i_4 = 0x16FFFC/4 \bmod 8192 = 0x1FFF$, $i_5 = 0xFFFFF8/4 \bmod 8192 = 0x1FFE$, $i_6 = 0xFFFFFC/4 \bmod 8192 = 0x1FFF$. Como há duas vias, ao acessar o i_1 e i_6 , é guardado na segunda via que até então está vazia, dessa forma não há substituições acessando as 7 sequências dadas.

c) Para forçar uma substituição de linha, é necessário ter um acesso à memória que o conjunto resultante do cache seja um com as duas vias ocupadas, nesse caso, após os 7 acessos, o conjunto $0x0$ está cheio. Dessa forma, pode-se acessar o endereço $0x320000$ que ao calcular o hash, $0x320000/4 \bmod 8192 = 0x0$, irá conflitar com o conjunto, sendo necessário a substituição de uma das vias. Para a seleção da via é usado então a política LRU (*least recently used*), ou seja, irá ser selecionado a via que faz mais tempo que foi acessada, nesse caso, irá substituir a via que continha os dados da sequência iniciada pelo endereço $0x000000$.