COMP40005 Computer Architecture Exam 2023

Imperial College London

May 10, 2023

1. (a) Is the number of instructions executed per second a reliable way of comparing performance of difference processors? Provide three reasons to support your answer.

   (b) A subset of the instructions for processor P can be sped up by n times when executing them on the accelerator A. When a program Q is compiled into the instructions of P such that a fraction f belongs to this subset, what is the speed improvement of P when it is accelerated by A?

   (c) It is found that the accelerator A costs c times as much as the processor P. What is the minimum fraction of instructions for programs that A has to speed up such that the accelerated processor is c times faster than P?

   (d) As technology advances, the performance of P is expected to improve by m times per month. How many months would pass when P alone can execute the program Q as fast as the current P when it is accelerated by A?

   (e) When a program R is compiled into the instructions of P, a fraction f of the instructions can be sped up n times by the accelerator A as before, and additionally a further fraction g of the instructions can be sped up k times by another accleerator B. What is the speed improvement of P when it is accelerated by both A and B?

The five parts carry, respectively, 15%, 10%, 15%, 25%, 35% of the marks.

2. (a) Compute the offset of each field, the size of the structure and its alignment requirement for x86-64, per each of the following structure declarations:

```
struct d1 { short i; long j; int *k; short *l; };
struct d2 { char a[3]; char *b[4] };
struct d3 { struct d1 c; struct d2 e[2]; };
```

Suppose we have a pointer D3 type of `struct d3` and the address of `struct d3` is stored in register `%rdi`, and integer index i in `%rsi`, respectively. Write the assembly code line for the following pseudo code instructions:

i. Store the value `D3->c.k` to register result `%rax`.

ii. Store the value `D3->e[index].a` to register result `%rax`.

(b) Consider the following x86-64 optimised assembly function:

```
sum:
    pushq %rbx
    movq %rdi, %rdx
    movl %esi, %edi
    call randomise
    testl %eax, %eax
    je .L4
    movl %eax, %eax
    leaq (%rbx, %rax, 8), %rdx
    movl $0, %eax
L3:
    addq (%rbx), %rax
    addq $8, %rbx
    cmpq %rdx, %rbx
    jne .L3
.L1:
    popq %rbx
    ret
.L4
    movl $0, %eax
    jmp .L1
```

Fill in the blanks in the following `sum` function. You may only use the variable names `result`, `count`, `init` and `seed`, but not the register names.

```
long sum(long *init, int seed) {
    int result = ___;
    int count = ___;
    while (___) {
        ___;
        ___;
        ___;
    }
    return ___;
}
```

(c) A certain byte addressable memory system is 12 bits wide. Memory accesses are to 1-byte words. The cache is 2-way set associated (two lines), has 4 sets and the blocks are 4 bytes in size. The contents of the cache are listed below.

Index	Tag	Valid	B0	B1	B2	B3	Tag	Valid	B0	B1	B2	B3
0	12	1	A2	DF	E4	DD	05	1	25	CF	1D	F8
1	C5	0	25	23	E4	ED	47	0	5A	09	67	AD
2	F3	0	3B	F2	43	4D	42	1	DD	7E	34	AB
3	97	1	DD	FF	34	FC	BB	0	5B	AC	7B	DD

The first tag, valid bit and bytes $0-3$ correspond to line 0. The second tag, valid bit and bytes $0-3$ correspond to line 1.

Provide the format of `0x42B` and `0x474` addresses. Indicate the fields and bits that would be used to determine the block offset, set index and tag of each address in hexadecimal format. Also, indicate the cache entry for these two addresses, i.e. whether a cache hit or miss occurs in each of the two addresses. If a cache hit occurs, provide the cache byte returned.

In summary, for `0x42B` and `0x474` addresses provide

- the block offset bits, index bits and tag bits,
- whether it is a cache hit or a miss,
- the cache byte returned if it is a hit.

The two parts carry, respectively, 75% and 25% of the marks.