Introduction to Computer Architecture Answers

(Based on Iurii’s answers - if you spot any mistakes, feel free to correct them!)

1a) To multiply by 7, multiply by 8 then -1

shl R2 R1 3 : R2 = 8R1 = 8n : 1 cycle

sub R2 R2 R1 : R2 = R2 – R1 = 8R1 – R1 = 7R1 = 7n : 2 cycles

3 cycles – mult takes 4 cycles so this implementation is 4/3 = 1.33 times faster

1b)

shl R2 R1 3 : R2 = 8n

shl R1 R1 2 : R1 = 4n

add R2 R2 R1 : R2 = 12n

shl R1 R2 R1 : R1 = 48n

add R2 R2 R1 : R2 = 60n

shl R1 R2 4 : R1 = 960n

add R2 R2 R1 : R2 = 1020n

10 cycles. Mult takes 4 cycles so this implementation is 10/4 = 2.5 times slower

1c) Booth’s algorithm would go through C and examine pairs of adjacent bits

1020 = 0011 1111 11002

Add 0100 0000 00002

Subtract 0000 0000 01002

shl R2 R1 10 : R2 = 1024n

shl R1 R1 2 : R1 = 4n

sub R2 R2 R1: R2 = 1024n – 4n = 1020n

This implementation takes 4 cycles, which is exactly as fast as simply using the mult instruction

2ai)

|  |  |  |  |
| --- | --- | --- | --- |
| Assembly code | Type | Expression | Value |
| **movl 8(%rbx), %eax** | **int** | **N[2]** | **M[xN + 8]** |
| leaq 8(%rbx, %rdx, 4), %rax | int\* | N + i + 2 | xN + 4i + 8 |
| movl 12(%rbx, %rdx, 8), %eax | int | N[2i + 3] | M[xN + 8i + 12] |
| leaq 20(%rbx, %rdx, 4), %rax | int\* | N + i + 5 | xN + 4i + 20 |

2aii) What a fun question which could easily be sorted if we got a markscheme :D

short compute (short x, short y) {

if (**x <= 0**)

return **0**;

if (**y <= 0**)

return **x**;

if (**x <= y**)

**increment(&x, y)**; **// y is zero extended for some reason**

else **// so I assume it is passed as an int argument**

**x = x << y**;

return **x**;

}

2b) Question split into a few parts:

128MB memory means addresses are 27 bits long

32-byte block means 5-bit offset

1. Assuming “fully associative”

Offset = log2(32) = 5 bits (0-4)

Index = 0 (no bits as cache is fully associative – one large set)

Tag = 27 – 5 = 22 bits (5-26)

1. Direct-mapped

Offset = log2(32) = 5 bits (0-4)

Index = log2(2\*1024\*1024 / 32) = log2(65536) = 16 bits (5-20) (each line is its’ own set)

Tag = 27 – 16 – 5 = 6 bits (21-26)

1. 8-way set associative

Offset = log2(32) = 5 bits (0-4)

Index = log2(65536 / 8) = 13 bits (5-17) (65536 lines divided into sets of 8 lines each)

Tag = 27 – 13 – 5 = 9 bits (18-26)

Hit and miss rate:

Access 1:

addr = 0, block = 0, not yet in cache à Miss

Access 2:

addr = 32, block = 1, not yet in cache à Miss

Access 3:

addr = 8, block = 0, in cache à Hit

Access 4:

addr = 32, block = 1, in cache à Hit

Access 5:\

addr = 64, block = 2, not yet in cache à Miss

Overall 3 misses, 2 hits. Miss rate = 3 / (3 + 2) = 3/5 = 60%

Notes:

0 = 0…0

32 = 0…100000

8 = 0…1000

64 = 0…1000000

No values are long enough for the associativity of the cache to affect the result

1. The cache is big enough to fit everything that was loaded
2. Associativity of the cache does not affect the outcome (directly mapped lines are always available)