**Department of Computing Examinations — 2016–2017 Session**

**MODEL ANSWER and MARKING SCHEME**

Examiner *wl* | Paper Code C210 = E 2-13

Question 1 | Page 1 *out of* 1

Question labels in left margin | Mark allocations in right margin

1a Accumulator architecture: memory load and store and arithmetic involve a special register called the accumulator.
Benefit: do not need to have register field in instruction
Drawback: arithmetic instructions involve memory access so can be slow

Load-store architecture: special instructions for memory load and store from and to registers, and other instructions do not involve memory access.
Benefit: arithmetic instructions does not involve memory, can be fast
Drawback: instructions often need extra bits to indicate which register — 6

b i)

| | |
|---|---|
| load 0 | A = M[0] |
| add 1 | A = A + M[1] |
| store 2 | M[2] = A |
| loadc 9 | A = 9 |
| sub 2 | A = A - M[2] |
| jmpneg L1 | if A<0 then PC = L1 |
| STOP | END |
| L1: loadc 6 | A = 6 |
| add 2 | A = A + M[2] |
| store 2 | M[2] = A |
| STOP | END |

loadc, ifneg STOP do not involve operand fetch.

ii) 10 instruction fetch, 6 operand fetch, 16 memory accesses — 7

c) i)

```
load R0 0
load R1 1
add R0 R0 R1
loadc R1 9
sub R1 R1 R0
jmpneg R1 L1
L0: STORE R0 2
    STOP
L1: loadc R1 6
    add R0 R0 R1
    jump L0
```

```
R0 = M[0]
R1 = M[1]
R0 = R0 + R1
R1 = 9
R1 = R1 - R0
if R1 < 0 then PC = L1
M[2] = R0
END
R1 = 6
R0 = R0 + R1
PC = L0
```

ii) 11 instruction fetch, 3 operand fetch, 14 memory accesses — 7

2a i) Under what conditions would a load instruction cause a write to memory?

*If the cache is using a write-back policy; and the block being read is not in the cache; and the block being evicted is dirty; then the dirty block will need to be written to memory.*

**Marks:** 2

ii) A non-pipelined processor is to be converted to a pipelined processor with two stages. Give upper and lower bounds on the pipelined processor's clock rate, and briefly comment on how realistic those bounds are.

*The worst-case is that all logic ends up in one stage, which means the critical path does not change, and the clock-rate stays the same The best-case is that the critical path is exactly split between both stages, so the clock-rate is 2x higher. So the relative clock rate is between 1x and 2x that of the original.*

*In reality there will likely be some reduction in low-level features such as clock quality (e.g. accuracy of duty cycle), clock distribution (as it needs to reach more FFs), and flip-flop setup and hold time. This means the upper bound is more like $(2 - \epsilon)$, where $\epsilon$ is circuit and process dependent.*

*The lower bound is extremely conservative, as it is only likely to happen where the entire critical path can be attributed to one single critical component. Any single cycle processor will contain multiple gates in the data path, so there is likely to be some some non-trivial component that can be moved into a different cycle.*

**Marks:** 3

iii) Caches can be inserted before or after physical to virtual translation. Give one advantage of each approach.

*Doing translation before the cache means that shared physical pages can be shared by multiple processes in the cache. Doing translation afterwards means that the TLB does not need to be accessed before going to the cache.*

**Marks:** 2

b Most contemporary CPUs satisfy all three of these properties:

- A : They are pipelined
- B : There is a shared instruction and data address space
- C : They have separate instruction and data caches

i) Why do properties A and B imply property C?

*In a pipelined processor, one instruction may be attempting to fetch at the same time as another is trying to read or write. As there is only a single memory, this would cause a structural hazard, and cause instructions to stall when another instruction was reading/writing. This is resolved by inserting separate instruction and data caches.*

**Marks:** 3

ii) Even when property A does not hold for a CPU, it is common for property C to still hold. Give two reasons why is it still useful to have separate instruction and data caches in a non-pipelined processor

*Instructions and data often live in different parts of the address space, so the instruction and data caches will tend to hold different sets of information. The instruction and data cache may also benefit from different associativity and block sizes, due to the different access patterns. It may be useful to optimise the two caches differently for hit-time versus capacity.*

**Marks:** 3

c Assume a re-useable cache design which is parametrised by cache capacity $n$. Extensive modelling has resulted in the following scaling properties for a given capacity (where $c_a$, $c_h$, and $c_m$ are silicon-technology specific constants):

- Area: $a(n) = c_a\, n$
- Hit time : $h(n) = c_h \log_2 n.$
- Miss rate : $m(n) = 1/(c_m\, n)$

Two instances of the generic cache will be used to create separate data and instruction caches for a pipelined processor.

i) Explain why each of the three modelled properties are plausible, in terms of the construction and behaviour of caches. Use sketches if necessary.

*Area is proportional to capacity, so doubling n should roughly double the area. Hit time is related to the number of places that need to be searched, and can be implemented as a multiplexor tree, and multiplexor tree depth is logarithmic, so search time is also logarithmic. Miss rates go down as cache size increases, and it is plausible that doubling the cache size will halve the miss rate.*

**Marks:** 4

ii) Given a total area budget of $b$ for both the instruction and data cache, what is the optimal balance between the capacity of the instruction cache ($n_i$) and the data cache ($n_d$) in terms of cycle time?

*Area constraint is $b = c_a + (n_d + n_i)$ We have best cycle time when both $h(n_i)$ and $h(n_d)$ are minimised, so we should choose $n_i = n_d$. (Another argument is that the minimum cycle time is achieved when $n_i$ and $n_d$ are zero, but that is a degenerate solution.)*

**Marks:** 3

*The three parts carry, respectively, 35%, 30%, and 35% of the marks.*

**Department of Computing Examinations — 2016–2017 Session**

**MODEL ANSWER and MARKING SCHEME**

Examiner *wl* | Paper Code C210 = E 2·13

Question 1 | Page 1 *out of* 1

Question labels in left margin | Mark allocations in right margin

1a Accumulator architecture: memory load and store and arithmetic involve a special register called the accumulator.
Benefit: do not need to have register field in instruction
Drawback: arithmetic instructions involve memory access so can be slow

Load-store architecture: special instructions for memory load and store from and to registers, and other instructions do not involve memory access.
Benefit: arithmetic instructions does not involve memory, can be fast
Drawback: instructions often need extra bits to indicate which register — 6

b i)

| | |
|---|---|
| load 0 | A = M[0] |
| add 1 | A = A + M[1] |
| store 2 | M[2] = A |
| loadc 9 | A = 9 |
| Sub 2 | A = A − M[2] |
| jmpneg L1 | if A<0 then PC = L1 |
| STOP | END |
| L1: loadc 6 | A = 6 |
| add 2 | A = A + M[2] |
| store 2 | M[2] = A |
| STOP | END |

loadc, ifneg STOP do not involve operand fetch

ii) 10 instruction fetch, 6 operand fetch, 16 memory accesses — 7

c) i)

```
    load R0 0
    load R1 1
    add R0 R0 R1
    loadc R1 9
    Sub R1 R1 R0
    jmpneg R1 L1
L0: STORE R0 2
    STOP
L1: loadc R1 6
    add R0 R0 R1
    Jump L0
```

```
R0 = M[0]
R1 = M[1]
R0 = R0 + R1
R1 = 9
R1 = R1 − R0
if R1 < 0 then PC = L1
M[2] = R0
END
R1 = 6
R0 = R0 + R1
PC = L0
```

ii) 11 instruction fetch, 3 operand fetch, 14 memory accesses — 7

2a i) Under what conditions would a load instruction cause a write to memory?

*If the cache is using a write-back policy; and the block being read is not in the cache; and the block being evicted is dirty; then the dirty block will need to be written to memory.*

**Marks:** 2

ii) A non-pipelined processor is to be converted to a pipelined processor with 2 stages. Give upper and lower bounds on the new per-instruction latency, relative to the non-pipelined processor.

*The worst-case is that all logic ends up in one stage, which means the clock-rate stays the same, but each instruction takes 2 cycles. The best-case is that the critical path is exactly split between all 2 stages, so the clock-rate is 2x higher, and each instruction has the same latency. So the relative latency is between 1x and 2x that of the original.*

*Concerns like setup/hold time could be introduced, which add to the worst-case delay; however they are not needed for this level of calculation.*

**Marks:** 3

iii) Caches can be inserted before or after physical to virtual translation. Give one advantage of each approach.

*Doing translation before the cache means that shared physical pages can be shared by multiple processes in the cache. Doing translation afterwards means that the TLB does not need to be accessed before going to the cache.*

**Marks:** 2

b Most contemporary CPUs satisfy all three of these properties:

- A : They are pipelined
- B : There is a shared instruction and data address space
- C : They have separate instruction and data caches

i) Why do properties A and B imply property C?

*In a pipelined processor, one instruction may be attempting to fetch at the same time as another is trying to read or write. As there is only a single memory, this would cause a structural hazard, and cause instructions to stall when another*

*instruction was reading/writing. This is resolved by inserting separate instruction and data caches.*

**Marks:** 3

ii) Even when property A does not hold for a CPU, it is common for property C to still hold. Give two reasons why is it still useful to have separate instruction and data caches in a non-pipelined processor

*Instructions and data often live in different parts of the address space, so the instruction and data caches will tend to hold different sets of information. The instruction and data cache may also benefit from different associativity and block sizes, due to the different access patterns. It may be useful to optimise the two caches differently for hit-time versus capacity.*

**Marks:** 3

c Assume a re-useable cache which can be parametrised for a given system with a capacity $n > 1$. The cache has been modelled and found to have the following properties for a given $n$:

- Area: $a(n) = c_a n$
- Hit time : $h(n) = c_h \log_2 n$.
- Miss rate : $m(n) = 1/(c_m n)$

The cache will be used to create a data cache and instruction cache for a pipelined processor.

i) Explain why each of the three modelled properties are plausible, in terms of the construction and behaviour of caches. Use sketches if necessary.

*Area is proportional to capacity, so doubling $n$ should roughly double the area. Hit time is related to the number of places that need to be searched, and can be implemented as a multiplexor tree, and multiplexor tree depth is logarithmic, so search time is also logarithmic. Miss rates go down as cache size increases, and it is plausible that doubling the cache size will halve the miss rate.*

**Marks:** 4

ii) For a combined instruction and data cache area budget of $a$, what is the optimal balance between the capacity of the instruction cache ($n_i$) and the data cache ($n_d$) in terms of cycle time?

*Area constraint is $a = c_a + (n_d + n_i)$ Have best cycle time when both $h(n_i)$ and $h(n_d)$ are minimised, so we should choose $n_i = n_d$. (Another argument is that the minimum cycle time is achieved when $n_i$ and $n_d$ are zero, but that is a degenerate solution.)*

**Marks:** 3

*The three parts carry, respectively, 35%, 30%, and 35% of the marks.*