EGT3
ENGINEERING TRIPOS PART IIB

Monday 29 April 2019 2 to 3.40

**Module 4F14**

**COMPUTER SYSTEMS**

*Answer not more than **two** questions.*

*All questions carry the same number of marks.*

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

*Write your candidate number **not** your name on the cover sheet.*

**STATIONERY REQUIREMENTS**
Single-sided script paper

**SPECIAL REQUIREMENTS TO BE SUPPLIED FOR THIS EXAM**
CUED approved calculator allowed
Engineering Data Book

**10 minutes reading time is allowed for this paper at the start of the exam.**

**You may not start to read the questions printed on the subsequent pages of this question paper until instructed to do so.**

1 (a) Explain what is meant by a *pipelined* datapath. What are pipeline *hazards*? [20%]

(b) The following extract of MIPS code increments all the elements of an $n$-element array by the contents of `$10`. Each element of the array is four bytes. The starting address of the array is `Astart` and $4n$ is in `$11`. $n$ is a multiple of four.

```
      add $9,$0,$0        # clear $9 to zero
Loop: lw $8,Astart($9)    # $8 loaded with data at address $9+Astart
      add $8,$8,$10       # $8 loaded with $8+$10
      sw $8,Astart($9)    # $8 stored at address $9+Astart
      addi $9,$9,4        # $9 loaded with $9+4
      bne $9,$11,Loop     # jump back 4 instructions if $9≠$11
```

The code is run on the datapath in Fig. 1. There is no data forwarding. Data hazards are resolved by stalling. Branches are assumed to be not taken, with pipeline flushing if they are taken. Calculate the minimum number of clock cycles required to execute the code:

(i) as is; [10%]

(ii) after adding a data forwarding unit to the pipeline; [10%]

(iii) after unrolling the loop by a factor of four (i.e. looping $n/4$ times and incrementing four array elements each time around), and reordering/adjusting instructions as necessary; [20%]

(iv) after making the pipeline 2-way superscalar, and reordering/adjusting instructions as necessary, with a restriction that only one load/store instruction can be issued at a time. [20%]

Each enhancement builds on the earlier ones, so you need to consider each new enhancement alongside all the previous ones. Be sure to justify your answer in each case.

(c) In (b), you calculated the *minimum* number of clock cycles required. Why might the actual number be higher? Which of the enhancements in (b) is most likely to increase the difference between the minimum and actual numbers? Justify your answer. [20%]

(cont.

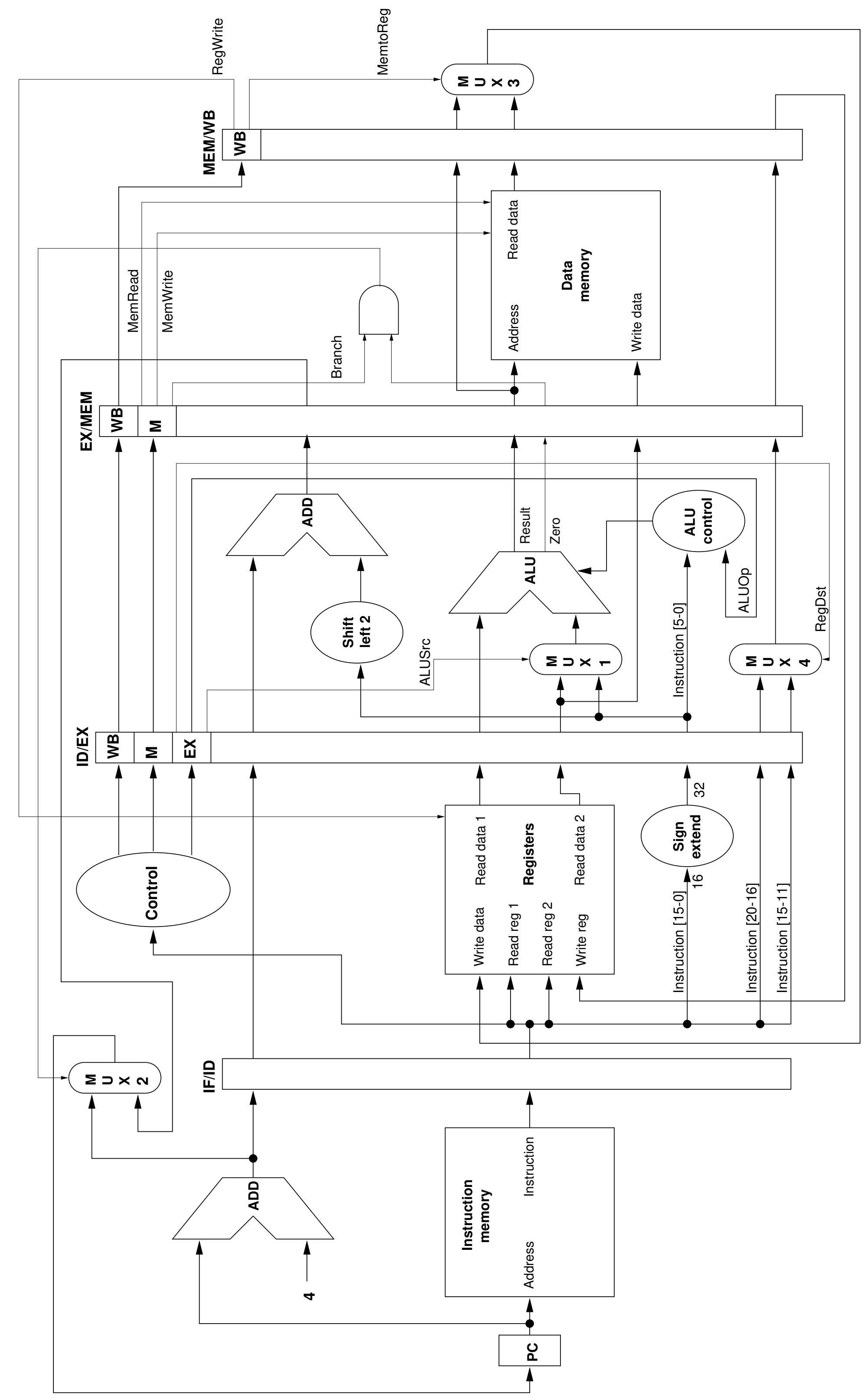

Fig. 1

(TURN OVER

2 (a) Explain why the inclusion of a cache, between the CPU and main memory, generally improves a computer's performance. [10%]

(b) Discuss briefly the relative advantages and disadvantages of direct mapped and set-associative caches. [15%]

(c) For a set-associative cache, sketch graphs showing how the miss rate and the miss penalty vary with (i) the block size and (ii) the degree of associativity. [25%]

(d) Figure 2 contains the obvious C++ code to multiply two $1000 \times 1000$ matrices together. The `int` matrix `c` is multiplied by the `int` matrix `b`, and the result stored in the `int` matrix `a`. Figure 3 contains an alternative version of the code, which uses a technique called *blocking* to reduce the cache miss rate: in this example the *blocking factor* is 10. Consider running the two code segments on a machine with a 1 KByte fully-associative data cache, 1-word blocks and least recently used (LRU) block replacement.

(i) If an `int` requires 4 bytes of storage, how many matrix elements can fit in the data cache? [5%]

(ii) Estimate the total number of data cache misses for the version of the code in Fig. 2. [20%]

(iii) Estimate the total number of data cache misses for the version of the code in Fig. 3. [25%]

```
for (i=0; i<1000; i++)
  for (j=0; j<1000; j++)
    a[i][j] = 0.0;

for (i=0; i<1000; i++)
  for (j=0; j<1000; j++)
    for (k=0; k<1000; k++)
      a[i][j] += b[i][k] * c[k][j];
```

Fig. 2

```
for (i=0; i<1000; i++)
  for (j=0; j<1000; j++)
    a[i][j] = 0.0;

blockingFactor = 10;
jj = 0;
kk = 0;
while (jj < 1000) {
  while (kk < 1000) {
    for (i=0; i < 1000; i++)
      for (j=jj; j < jj + blockingFactor; j++) {
        r = 0;
        for (k=kk; k < kk + blockingFactor; k++)
          r += b[i][k] * c[k][j];
        a[i][j] += r;
      }
    kk += blockingFactor;
  }
  kk = 0;
  jj += blockingFactor;
}
```

Fig. 3

3 (a) Explain why the latency of the arithmetic logic unit (ALU) is of paramount importance in computer hardware design. [10%]

(b) Describe the operation of a single level, 4-bit carry-lookahead adder. [20%]

(c) Figure 4 shows a *m*-bit ripple-carry adder with one modification. The final carry out, cm, is selectable between either the standard ripple-carry signal or the initial carry-in signal c0. The multiplexor is controlled by the combinatorial logic block CL, which distinguishes between two types of addition by examining the bits of a and b.

(i) An example of a Type 1 addition is 0100 + 1011 + c0, while examples of Type 2 additions are 0100 + 1001 + c0 and 0110 + 1011 + c0. By working through these examples by hand, and generalising, explain why for Type 1 additions cm is always equal to c0, whereas for Type 2 additions cm is always independent of c0. Deduce an expression for the combinatorial logic inside the block CL. [20%]

(ii) *k* of the circuits in Fig. 4 are chained together, by connecting the cm output of one to the c0 input of the next, to form a *n*-bit adder ($n = km$) called a *block-carry-skip adder*. If the latencies of the full adders, the CL block and the multiplexor are all $T$, derive an expression for the latency of the *n*-bit block-carry-skip adder. Hence, deduce the optimal block size *m* in terms of *n*. [40%]

(iii) Discuss briefly the relative advantages and disadvantages of block-carry-skip adders and carry-lookahead adders. [10%]

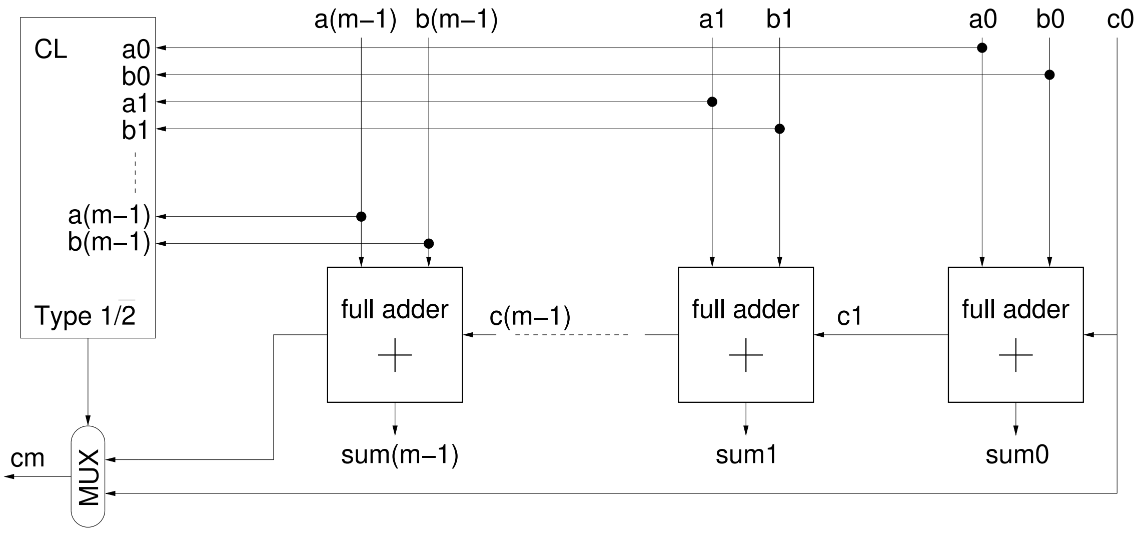

Fig. 4

**END OF PAPER**

# Part IIB 2019

# Module 4F14: Computer Systems

# Numerical Answers

1. (b) (i) $17n + 5$ (ii) $9n + 2$ (iii) $17n/4 + 2$ (iv) $11n/4 + 2$
2. (d) (i) $256$ (ii) $\sim 2 \times 10^9$ (iii) $\sim 2 \times 10^8$
3. (c) (i) Type 1 = (a0 $\oplus$ b0) . (a1 $\oplus$ b1) . . . (a(m-1) $\oplus$ b(m-1))

   (ii) Overall latency is $(2m + k)T$, optimal when $m = \sqrt{n/2}$