UNIVERSITY OF EDINBURGH

FACULTY OF SCIENCE AND ENGINEERING

COMPUTER SCIENCE THREE

COMPUTER ARCHITECTURE

Wednesday 5 June 2002

09:30 to 11:00

Examiners:

Prof. K. Turner (External)

Prof. C. Stirling (Chairman)

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

1. You are hired by ED Computers to design their new processor. You are told that this processor is to be used mainly for scientific programming and that the core of the workload consists of vector and matrix operations. In particular, vector scaling ($x = x * y + c$, where: x and y are vectors and c is a scalar) is the single most important operation to be executed by this machine. Your job is to design the best possible processor architecture for this operation. Assume the following code for the vector scaling operation:

```
for (i=0; i<N; i++)
  x[i] = x[i]*y[i]+c;
```

(a) Write the assembly code for this loop using the DLX instruction set. Schedule your assembly code as well as you can for the standard DLX pipeline with forwarding. You can move instructions within the same iteration, but do not unroll the loop and do not use the branch delay slot. Assume the following:

- 5-stage pipeline: F, D, X, M, W
- double precision floating point array elements and scalars (8 bytes)
- the value of N is already stored in r1
- the base addresses for x and y are already stored in r2 and r3, respectively
- the value of c is already stored in f1
- FP add and multiply take 4 cycles in the X stage and are fully pipelined

How many cycles does the whole loop take as a function of N? [*8 marks*]

(b) Unroll the loop as many times as necessary to eliminate stalls on the same pipeline as part (a). How many cycles does the whole loop take now? [*7 marks*]

(c) Using the unrolled loop from part (b), schedule your code on a superscalar version of DLX that has two functional units of each integer/memory and FP. This superscalar processor can, therefore, issue any two instructions per cycle. How many cycles does the loop take now? [*5 marks*]

(d) Some processors designed for scientific processing have used a combined multiply-add instruction. This instruction is of the form:

```
multadd f1, f2, f3
```

and will do: f1=f1*f2+f3. You are then asked to investigate the potential benefits of adding such an instruction to the processor. Your VLSI design team tell you that they can implement this operation using the same 4 cycles in the X stage as the other FP instructions, but the clock time will have to be stretched by 10%.

QUESTION CONTINUES ON NEXT PAGE

Change the assembly code for the non-unrolled loop of part (a), with the addition of the multadd instruction. How many cycles does the whole loop take now? Assuming that latency for the remaining instructions is not changed, which machine will be faster, with or without the multadd instruction, and by how much? [*5 marks*]

2. (a) Explain the principle of locality and how it relates to the idea of a memory hierarchy. [*5 marks*]

   (b) Describe the three types of cache misses. [*5 marks*]

   (c) Describe at least two techniques to reduce *each* of: cache miss rate, cache miss penalty, cache hit time. [*10 marks*]

   (d) Discuss the advantages and disadvantages of both physical and virtual address caches. [*5 marks*]

3. (a) Explain the behaviour of a 2-bit saturating counter branch predictor. Show the state of the predictor and the transition for each outcome of the branch. *[5 marks]*

(b) Consider the following code:

```
for (i=0; i<N; i++)
  if (x[i] == 0)
    y[i] = 0.0;
  else
    y[i] = y[i]/x[i];
```

Assume that the assembly code generated is then:

```
loop: ld   f1, 0(r2)
      ld   f2, 0(r3)
      bnez f1, else
      addd f2, f0, f0
      bez  r0, fall
else: divd f2, f2, f1
fall: addi r2, r2, 8
      addi r3, r3, 8
      subi r1, r1, 1
      sd   -8(r3), f2
      bnez r1, loop
```

where:

- the value of N is already stored in r1
- the base addresses for x and y are stored in r2 and r3, respectively
- register f0 contains the value 0

Assuming that every other element of x has the value zero, starting with the first one, show the outcomes of predictions when a 2-bit saturating counter is used to predict the inner branch *bnez f1, else*. Assume that the initial value of the counter is 00. *[5 marks]*

(c) Again, if every other element of x has the value zero, starting with the first one, show how the code of part (b) will be executed on the standard DLX pipeline with forwarding, if the processor uses a 2-bit saturating counter branch predictor to predict the outcome of the inner branch *bnez f1, else*. Assume that the other branches (*bez r0, fall*; *bnez r1, loop*) are always predicted correctly. Do not use the branch delay slot. Further assume the following:

QUESTION CONTINUES ON NEXT PAGE

- 5-stage pipeline: F, D, X, M, W
- FP divide takes 4 cycles in the X stage and is fully pipelined
- a branch prediction occurs during the D stage and the branch is finally resolved during the M stage
- the target instruction address is also known once a prediction is made
- during the prediction the follow up instruction is fetched, if the prediction is *not taken* then the follow up instruction continues, if the prediction is *taken* then it is squashed
- when the branch is finally resolved, incorrectly predicted instructions are squashed

[*5 marks*]

(d) Rewrite the assembly code of part (b) and eliminate all the internal branches (*bnez f1, else*; *bez r0, fall*) with predicated execution. Assume the following instructions are available for predication:

- `seqp px, fx, fy`

  where px is any one of 8 predicate registers (p0-p7), fx and fy are any one of 32 floating point registers (f0-f31). This instruction sets the predicate register px to TRUE iff fx=fy
- predicated versions of all ALU floating point operations:

  `adddp fx, fy, fz, px; divdp fx, fy, fz, px`

  These instructions execute iff predicate is TRUE.

  `adddp fx, fy, fz, ^px; divdp fx, fy, fz, ^px`

  These instructions execute iff predicate is FALSE.
- predicate registers are computed in the X stage and a dependence on a predicate register causes a stall until the value is computed in X
- seqp takes 1 cycle at the X stage; adddp and divdp take 4 cycles at the X stage

[*5 marks*]

(e) Show the execution of the predicated code of part (d) in the processor pipeline. How fast is the predicated code compared to the original code with branch prediction? [*5 marks*]