Test 1: Lectures 2 to 9

EEE4120F 2020-03-10

SOLUTIONS

Question 1: Easy Multiple Choice **[25 Total]**

1.1 Let's start with something easy: The issue of a "golden measure" is discussed at various points in this course… what does this refer to, select the option that best fits:

(a) The pretty solution, usually in MATLAB, that takes as little memory as possible.
(b) The fast solution that you are unlikely to beat with you parallel solution.
(c) The golden measure is the strategy of weighting your various solutions to the problem, the golden solution is the heaviest of your solutions.
(d) The golden measure may run slowly, it might not be optimized, but you know it gives numerically correct results. ←
(e) The golden measure is the final version of your development effort, which you deliver.

[5 marks]

1.2 A speed-up graph typically plots…

(a) Speed-up (vertical axis) versus memory utilization (on horizonal axis).
(b) Speed-up (vertical axis) versus processing elements (on horizonal axis). ←
(c) Speed-up (vertical axis) versus run time (on horizonal axis).
(d) Speed-up (vertical axis) versus programming complexity (on horizonal axis).
(e) Speed-up (vertical axis) versus lines of code (on horizonal axis).

[5 marks]

1.3 What is a datapath, select the most accurate definition below…

(a) Processing pieces that save data in a computer architecture.
(b) Functional parts that transfer data from one latch to another.
(c) Pieces of the computer system that transform the bits into bytes.
(d) Processing pathways that move operations through the computer.
(e) Functional units that carry out data processing operations for a computer system. ←

[5 marks]

1.4 The classic von Neumann computer has four main pieces, these are…

(a) I/O, Logic Components, Controller, Arithmetic Component.
(b) Controller, Astrographic Logical Unit, Storage, I/O.
(c) Memory, Control Unit, ALU, I/O. ←
(d) Microcontroller, RAM, program ROM, I/O controller.
(e) ALU, Control, Logic, Connections.

[5 marks]

1.5 In the slides it discusses an OpenCL kernel, a kernel is something that (choose the most correct option below) …

(a) The kernel is typically a fairly small, but highly parallelized piece of code that runs on the accelerator. ←
(b) The kernel comprises the main body of code of an OpenCL-enabled program.
(c) The kernel code has many limits place on it, such as being able to use a small number of registers.
(d) The kernel is a microcoding solution whereby the main processor can offload tasks to the few but very fast microcode instructions of the microcode state machine.
(e) The kernel is the processing hardware on which the OpenCL code runs.

[5 marks]

Note: *For Questions 2 and 3 Please refer to LLPM Processor Architecture presented in Appendix A (can detached the page!)*

Please also read the notes at bottom of Appendix A about the two-layer writeback operation.

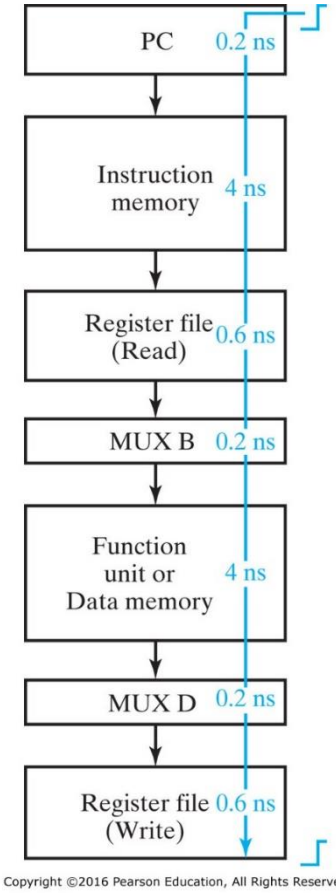
Question 2: Microcoding task **[25 Total]**

Answer all following questions…

2.1 Let us consider processors in general (the LLPM being an instance). Usually, processors have a certain 'delay path' between loading an instruction pointed to by the program counter (PC) and incrementing the PC, and writing back register results after the instruction has completed.

(a) Briefly discuss what sort of other stages a processor typically has and the sequence of these (you've already been given the first and last stages: reading the instruction and incrementing the PC, and writing back to the register file). [6 marks]

A: The stages are typically divided, at the high level, into Fetch, Decode, and Execute – but by mentioning the delay path and more specifics of instruction loading and PC incrementing I was hoping to see more consideration for the types of operations carried out. You could get max 3 marks for just mentioning these 'big 3' categories. But I was seeking more insight, your deeper understanding of what is involved, following the structure that was discussed in lecture 5, shown on the right. In other words, indication of the stages of accessing instruction memory (and possibly incrementing PC), reading the register file, connecting the register requested to intermediate registers (e.g. inputs to the ALU), executing the ALU function, moving data from the ALU to where the results needs to go (maybe writing to a memory address), and then writing back to the register file if needed. This aspect of moving register values around is significant to consideration for data paths.



Worst-case delay path in a single-cycle computer design

(b) Consider that you have estimated the maximum delay path duration (i.e. how long to complete the longest instruction) for a simple single cycle processor. Do you think it is better to make the clock period for the processor 20% longer, or 20% shorter, than the maximum delay path duration? Provide a brief motivation for your answer. [4 marks]

A: This was a bit obvious. It's of course safer to make it 20% longer, to account for things such as ensuring there is enough time for the signals to settle before the next clock, to allow for the clock signals to possibly propagate slower operating at high temperatures, e.g. if the chip starts heating up, it might take slightly longer for signals to move around within the chip, but clock (if it has an external clock, which wasn't indicated, might still be arriving at the same rate; i.e. there can be a temperature discrepancy the board remains relatively cool but the chip gets relatively hot, so your clock might be arriving at a regular *x* MHz to the chip, but it is not getting as perfectly propagated within the chip, and this would be especially so for a softcore processor, where the CPU is implemented using a FPGA which has a complex switching fabric to content with).

2.2 Here is example of LLPM microprogram to implement the NEG instruction. This example aims to illustrate how the microinstructions listed on the previous page can implement the more advice machine instructions for the LLPM (comments are given after the ';').

NEG R1:

```
SelF  <= 4    ; set F to -1
SelM  <= 0    ; set G to F
SelLb <= 1    ; loop back G value, A = G = -1
SelB  <= 0    ; B = R1
SelBS <= 0    ; B = B  (as opposed to -B, which would cancel out the -1)
SelOp <= 3    ; Activate A MUL B  i.e. -1 x R1
SelF  <= 3    ; F = A MUL B
SelM  <= 0    ; G = F
SelRw <= 0    ; R1 = F  (i.e. R1 now stores -1 x old value of R1)
```

TODO: Now it's you're turn... do the following:

Write the microcode that will implement the machine instruction:

SHL R1, #2 (i.e. shift left R1 by 2 bits)

A: One of the reasons for this question was to make it partly educational. The 'loop back' type operation Is sometimes used for processors that are highly resource constrained, where one both wants to squeeze in CISC-like instructions, to save on code memory, but also sacrifice performance by reusing circuitry. This is of course just a very simple example of such a technique, where were are reusing the single-bit shift register in order to implement a multiple-bit shifts without needing to add additional instructions. In this SHL instruction the 'N' operand would have been set to 2 … but I did not specifically state that in the question because I did not want to over-complicate the issue (which is of course a common problem in designing processor architecture); so it's find just to hard-code shifting by 2. Now that you know why you were given this problem, let's solve it…

A:

SHL R1, #2:

```
SelMP <= 0    ; set MP = MP, just for safety, no marks for this line.
SelA  <= 0    ; select R1 which will be fed to A
SelLb <= 0    ; set A to R2
// note: we don't care what B is, so best not to waste time on it
// PS: could do cin <= 0 if you don't want rotate left, but let's not bother.
SelOp <= 0    ; enable SHL
SelF  <= 0    ; F = A SHL 1
SelM  <= 0    ; G = F
SelLb <= 1    ; A = G
SelOp <= 0    ; enable SHL
   -- it's OK if you skip the above SelOp line, assuming it is change triggered
   -- but in reality one would probably need to toggle a clock line or change a
   -- flipflip set line from low to high and back to low, which you could assume
   -- as being past of the process for assigning a register to a value.
   -- For simplicity sake, I did not add these set/clear lines registers.
SelF  <= 0    ; F = A SHL 1
SelM  <= 0    ; G = F
SelLb <= 0    ; cancel loopback (in reality would be before SelF <= 0)
SelRw <= 0    ; R1 = G
// DONE! Value that was in R1 has been shifted left twice
```

Logical and clear: up to 5 marks rewarded

[15 marks]

Question 3: Microcoding and BCE Considerations **[30 Total]**

Consider that the LLPM machine structures are design in the following way:

OPCODE (6-bits)	N (4-bits)	Rx (2 bits)	Ry (2 bits)	Rz (2 bits)

(Note: N is bits for constant values, e.g. number of bits, the #*n*, to shift for the SHL Rx,#n instruction)

3.1 Suggest a strategy by which you could implement the microprogram interpreter, i.e. the hardware mechanism to run microprograms (this would be combinational logic) – I'm referring to this as the *microrunner* module below…

```
module microrunner (
    input [5:0] opcode,
    input [3:0] N,
    input [1:0] Rx,
    input [1:0] Ry,
    input [1:0] Rz,
    output done );
```

[Total for question 3.1: 15 marks]

A: For the answer to this question, I'm expecting some discussion about e.g. a statemachine that will have the microprograms in ROM, the OPCODE would indicate the starting state of the statemachine. Additionally, or alternatively, there could be a lookup table that translates the OPCODE into a starting point in a dictionary of microprograms, each line in the microprogram could invoke a particular state or operation, with the last line being a return (e.g. back to loading next machine instruction). The marking is based on the logic of the explanation and suggested approach.

3.2 Now for the moment for which you have been **SHIVERING IN ANTICI...PATION!**... In order words, The BCE Question!...

Let us assume there are two versions of the LLPM, the non-fancy baseline version which we can consider as the 1-BCE. Then there is the more impressive and substantive LLPM-C, the CISC version of the LLPM which takes a massive 6-BCE worth of resources to implement. The speedup performance of the LLPM-C over the LLPM is 3, i.e. perf(6) = 3. This on its own is pretty impressive (as opposed to being sqrt(6)).

Let us consider that we are developing an asymmetric **mutli-processor chip that has 8x BCEs** worth of resources (i.e. **n=8**). We have decided to consume 6 of these for just one LLPM-C leaving 2 standard LLPMs. i.e. we have 1x6-BCE and 2x1-BCEs.

(a) What is the asymmetric speedup of the processor containing 1x6-BCE + 2x1-BCEs over a simple 1-BCE processor? You can assume the 6-BCE megacore will run the sequential code. The code considered has 20% sequential and 80% parallel (assume all the available cores will run this part in separate threads). *[marks for 3.2(a): 5]*

A:

For the 1x1 BCE the speedup is 1 (by definition, in case you were wondering).

For asymmetric 1x6-BCE and 2x1-BCE, it would be…

Using equation:

$$\text{Asymmetric Speedup} = \frac{1}{\frac{1-F}{\text{perf}(R)} + \frac{F}{\text{perf}(R) + (n-R)}}$$

perf(R) =3 … lets call it: perfR = 3; n = 8; R = 6; F = 0.8 ; % 1-F = 0.2 (sequential)

So using OCTAVE:

speedup = 1 / ((1-F)/perfR + F/(perfR + (n-R)))

= **4.4118**

Wow, a speedup of 4.4. This is looking pretty good considering use of 3 processors for F part.

(b) If we consider the (unlikely) dynamic speedup scenario for the 8-BCE chip, where the 6-BCE megacore can magically swap between being a 6-BCE megacore and being 6x 1-BCE cores. Assume the program is still 20% sequential and 80% parallel.
(i) For this case, what is the speedup (over 1x 1-BCE) that can be expected if the 6-BCE runs the sequential part and the 8x 1-BCEs all run the parallel part. [5 marks]

A: For this we could use the dynamic multicore speedup:

$$\text{Dynamic Speedup} = \frac{1}{\frac{1-F}{\text{Perf}(R)} + \frac{F}{n}}$$

Again: perfR = 3; n = 8; R = 6; F = 0.8 ; % 1-F = 0.2 (sequential)

speedup = 1 / ((1-F)/perfR + F/n)

= **6**

Certainly looks pretty ideal, but probably overoptimistic.

(ii) Would this be better or worse than having 1x 6-BCE and 2x 1-BCEs running the parallel portion? (Motivate your answer to *ii*, e.g. including calculations). [5 marks]
[marks for 3.2(b): 10]

A:
To answer this question:
We are just looking at the parallel portion. So for the symmetric part we had:
parallel_speedup = (perfR + (n-R))
= 5 (i.e. parallel cores doing work)
It's pretty clear that this is doing to be the case if we are considering the type of averaged operation which leads to perf(6)=3. i.e. we have 2 basic cores + 1 x megacore 3x as fast 1 fancy = 5
For the dynamic core case, the parallel speedup is simply going to be 8, i.e. all 8 1-BCEs working together.
So, at face-value yes it looks like "this method" i.e. the dynamic multicore, would work better.

→ marking note: for this pretty obvious answer **you can get 4/5**, but if you show some more insight:

But there are other influencing factors….
Remember that Quiz2 worksheet about the B1 and B2 processors, i.e. the one that couldn't multiply (and who's programmers had to keep yelling at it work harder) and the other one that could multiply (and who's programmers told their friends how smart and well-behaved their processor was). Yes, that was about comparing two processors considering how some instructions might take a similar time between the two, the bulker one might do some instructions a bit faster, and there may be a few instructions that are massively (as in 100s) of times faster. The design can impact things a lot. We'll that's the point I was seeking …
i.e. Just assuming that the 1x 6-BCE will be worse than 6x 1-BCEs is an assumption, made when using these speedup calculates. We do not know enough about the description of these processors, such as the speed difference between individual instructions. Indeed the 6-BCE might even be implemented in such a way that certain operations are phenomenally faster than the other processor (e.g. maybe the 6-BCE can utilize multiple ALU elements at the same time, e.g. doing a SHL and SQUARE – i.e. B*B, at the same time – it looks like the basic LLPM architecture may already support that).
With a nice bit of though such as this you could **get the full 5/5**.

(Ok, the question could have been worded better to suggest this direction; but I didn't want to make it too obvious what I was trying to get at although if you'd done the worksheet you should have had a good idea of the limitations of using the speedup equations and the assumptions posed).

[Total for question 3.2: 15 marks]

Question 4: Short Explanations & and some post-BCE breathing space [20 Total]

To finish off here's a few quick questions…

4.1 If you were to run the same program on different PCs using different data would you be doing SPMD or MPMD? Chose which and explain the acronym.

A: SPMD, SPMD = Single Program Multiple Data

[5 marks]

4.2 What is the difference between "massively parallel" and "embarrassingly parallel"? Surely, they are the same thing… but if not briefly explain why they are not the same.

A: Massively parallel = 10,000 or more concurrent operations

Embarrassingly parallel = parallelization where little or no effort is required to separate the problem into parallel tasks. This is often the case where there exists no dependency and no need for communication between these parallel tasks.

[5 marks]

4.3 What are some of the parallel overheads that one is likely to encounter when developing parallel programs? (mention at least 3, the clarity of your explanation counts 2 marks.)

A: Parallel overhead: Amount of time to coordinate parallel tasks (excludes time doing useful work). Parallel overhead includes... operations such as: Task/co-processor start-up time, Synchronizations, communications, parallelization libraries (e.g., OpenMP, Pthreads.so), tools, operating system, task termination and clean-up time.

[5 marks]

4.4 Briefly explain what is meant by granularity in relation to solving computing problems? Would you say that the matrix operation A = A + 1 is fine-grained or course-grained, and explain briefly why.

A: Granularity = How big or small the parts are that the problem can be decomposed into, and/or how interrelated these sub-tasks are.

[5 marks]

END OF SOLUTIONS