Test 1: Lectures 2 to 9

EEE4120F 2020-03-10

Instructions:

- Make sure that your student number is on all your answer books.
- There are **4** questions, each divided into sub-questions. Answer all questions.
- Mark assignments: Q1: 25 marks, Q2: 25 marks, Q3: 30 marks!, Q4: 20 marks
- Total time: 1.5 h.
- Total marks: 100.
- *PS: there an amazing detachable cheatsheet on the last page!*

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]**

Microcoding is a means of interposes a layer of organization in the CPU hardware whereby a machine instruction is accomplished through the work of multiple smaller instructions called microinstructions, or the processing of "microcoding". Microcoding is heavily relied on by the LLPM processor design.

The registers at the top and at the bottom of the page (if you hold it in landscape) are registers that can be set via microcode, indeed these are the main operations available in the microcode instruction list. These are listed in the table below.

List of LLPM Microinstructions

Micro inst.	Explanation	Micro inst.	Explanation
SelRw <= *x*	Select general purpose register to assign to G (i.e. if *x*=0: set R1=G, if *x*=1, set R2=G, etc.).	SelF <= *x*	This is to select either a function output or value to send to F. e.g. if SelF=0, F=SHL result. If SelF=4, F=-1, if SelF=5, F=0, etc.
SelMP <= *x*	Update memory pointer (MP) register. If *x*=0 do nothing (i.e. MP=MP), If *x*=1 then MP=R1	SelM <= *x*	If *x*=0, then set G=F. If *x*=1 then read from memory pointed to by MP (i.e. G = Mem[MP])
SelA <= *x*	Select general purpose register to send to A (note: A only set to this register if SelLb=0)	SelB <= *x*	Select general purpose register to set to intermediate register B. (If *x*=1 then B=R2)
SelLb <= *x*	Select Loop-back: If *x*=1 then A=G else A= MuxA output	SelBS <= *x*	Select B swap sign, in order words negate B. e.g. If *x*=0 The current value of B is sent to ADD and MUL. If *x*=1 then NEG assumes B is in 2's complement and negates the sign of B, i.e. sends -B to ADD and MUL.
SelOp <= *x*	Chip select one of the functions. Note decoder before SelOp to ensure only one CS is high. E.g. if SelOp=0 then SHL is active, if SelOp=3 then MUL is active. etc.		
Note that when a register is assigned, e.g. SelA <= 0 immediately the MUX or device connected to that register will be triggered (e.g. SelA <=0 will immediately set the output of MUXA to R1)			

List of LLPM Machine Instructions (simplified listing)

Instruction	Explanation	Instruction	Explanation
NOP	Do nothing	NEG Rx	Rx = -Rx
CMP R1,R2	Compare R1 to R2 and set flags	CLR Rx	Rx = 0
ADD Rx,Ry,Rz	Rx = Ry + Rz	SNO Rx	Set Rx to -1
SUB Rx,Ry,Rz	Rx = Ry – Rz	SPO Rx	Set Rx to +1
MUL Rx,Ry,Rz	Rx = Ry x Rz	STW Rx	Set Rx to +2
SHL Rx,#*n*	Shift Left Rx by n bits (*n*=0 to 16)	LDR Rx,[Ry]	Rx = Mem[Ry]
SHR Rx,#*n*	Shift Right Rx by n (*n* = 0 to16)	STR Rx,[Ry]	Mem[Ry] = Rx
MOV Rx,Ry	Rx = Ry	JMP Rx	Jump to address Rx (i.e. PC = Rx)
MOV Rx,#*x*	Rx = constant value *x*	JMP*cnd* Rx	Conditional jump to address Rx
Note: Rx, Ry, Rz are place holders for any of the general purpose registers, i.e. R1, R2, R3, R4. In the conditional jump, *cnd* is Z for zero flag or equal, N for negative flag, P for positive flag, LT for less than, GT for greater than, LE for less than or equal, GE for greater than or equal.			
Note: The LLPM is a 16-bit processor, words are 2-bytes, instruction and memory addresses are 2-bytes.			

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]

(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]

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 apposed to -B)
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)

(Note: I mainly want to see your list of microcode operations to do this, as illustrated for the NEG instruction above; you are welcome to leave rough working or illustrations in your answer if you think that will better motivate your answer).

[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. Note that **you are *not required* to do any Verilog** for this question, you just need to **explain in English or pseudocode** the broad approach how you could go about implementing a microprogram interpreter. You may add an illustration as a means to help your explanation (this is not required). In your discussion you can assume there is an output bit (called 'done') that is set to 1 to tell the decode that the CPU is ready for the next instruction.

```
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]

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]*

(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]
(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]

[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.

[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.

[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.)

[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.

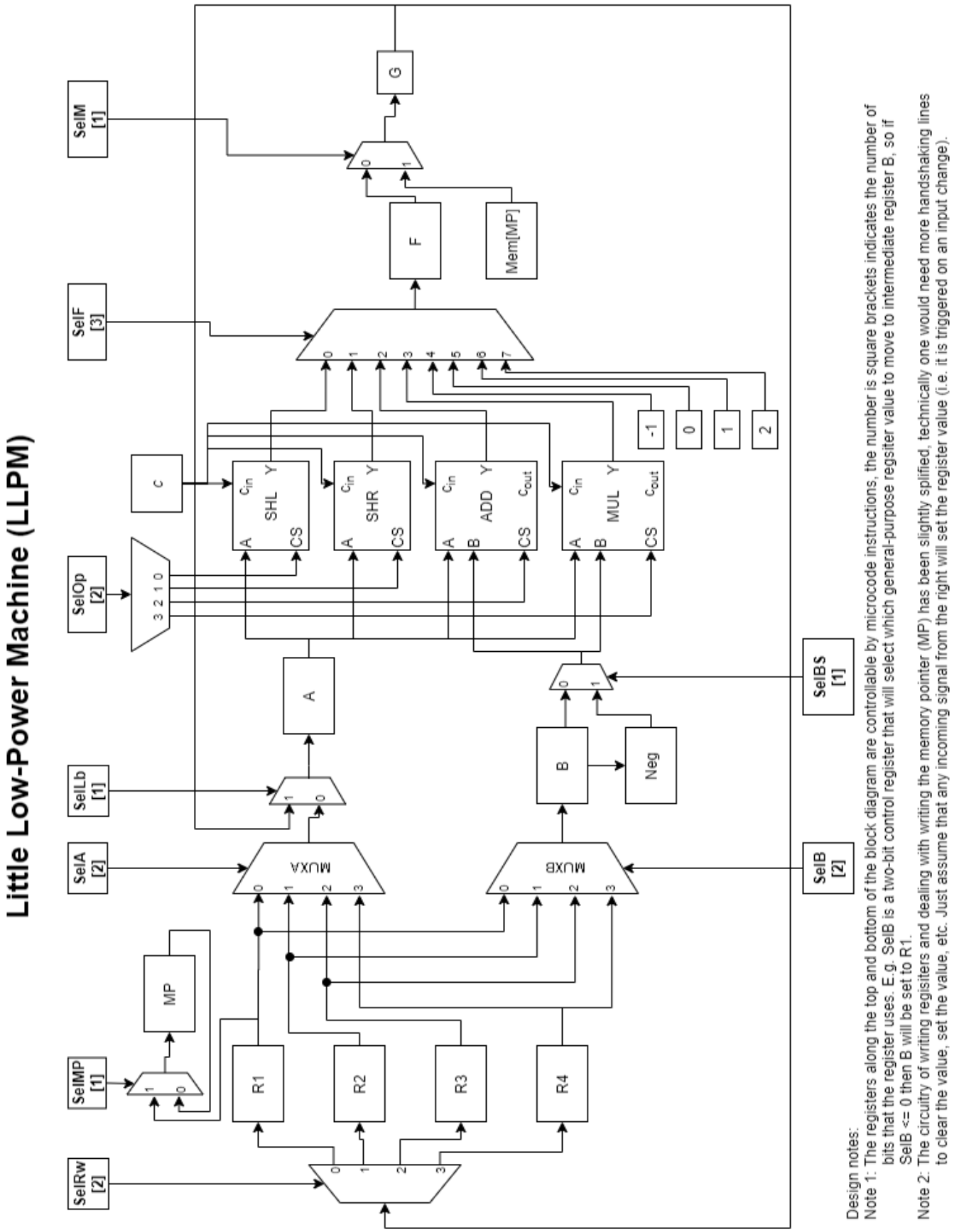
[5 marks]

CONGRATULATIONS,

YOU HAVE REACHED THE END OF THE TEST!

END OF TEST

APPENDIX A:
Little Low-Power Machine (LLPM)



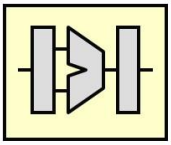
This is the architecture for the LLPM processor. You may want to detach this page, the back page of which has the cheatsheet. A key issue in this architecture is that there are two levels of writeback operation, the inner level (see SelLb) which writes a result immediately back to A, which can then go through the functions again, and SelRw which writes the result of the function executed (or memory access) back to a regsiter. Discarding F result (e.g. NOP) is just SelLb <= 1 and continue to next instruction.

CHEET SHEET

Amdahl's and Related Equations

speedup equation:	Sorry, you should really know this one by now!
enhanced section equation:	$\text{Speedup}_{\text{enhanced}}(f,S) = \frac{1}{(1-f) + \frac{f}{S}}$
n-core speedup equation:	$\text{Speedup}_{\text{parallel}}(f,n) = \frac{1}{(1-f) + \frac{f}{n}}$
Parallel Efficiency	$E = \frac{S}{p} = \frac{T_S}{pT_p}$

BCE Equations



symmetric speedup: (i.e. speedup of symmetric multicore over a 1-BCE)	$\text{Symmetric Speedup} = \frac{1}{\frac{1-F}{\text{perf}(R)} + \frac{F*R}{\text{perf}(R)*n}}$
asymmetric speedup: (i.e. speedup of asymmetric multicore over a 1-BCE)	$\text{Asymmetric Speedup} = \frac{1}{\frac{1-F}{\text{perf(R)}} + \frac{F}{\text{perf}(R) + (n-R)}}$
Dynamic multicore speedup: (i.e. speedup of dynamic multicore over a 1-BCE)	$\text{Dynamic Speedup} = \frac{1}{\frac{1-F}{\text{Perf}(R)} + \frac{F}{n}}$
Parameters explained:	n : number resources in 1-BCE cores available per chip R : number of 1-BCE cores a megacore cosumes in terms of resources (e.g. a 2-BCE core consumes twice as many resources as a 1-BCE, it *doesn't* mean it simply comprises two 1-BCE processing cores). F : parallel Fraction of execution 1-F : serial fraction of execution perf(1) = 1 = speedup of 1-BCE perf(R) = speedup of R-BCE over 1-BCE (note in some cases perf(R) can be less than 1 i.e. for cases the multicore is considered generally slower than the 1-BCE, but it doesn't mean to say it is slower for all processing jobs so make sure to read the question properly if you see this situation happening!).